Aalto University
School of Science
Department of Computer Science and Engineering

TIMO PUNKKA

Flexible New Product Development: Using Knowledge Transfer from Agile Software Development as a Catalyst for Adaptation
- Case Study and Systematic Literature Review

Licentiate Thesis submitted for official examination for the degree of Licentiate in Technology.

Espoo, December 11, 2015

Supervisor: Casper Lassenius

Instructor: Casper Lassenius

Aalto University
School of Science

Abstract of Licentiate Thesis

Author

Timo Punkka

Title of Thesis

Flexible New Product Development: Using Knowledge Transfer from Agile Software Development as a Catalyst for Adaptation – Case Study and Systematic Literature Review

Abstract

The life cycle of products is getting shorter and product development projects are more than ever challenged by frequent change. In response to this evolution, Agile software development is gaining a foothold in the software industry. Agile software development relies on iterations, collaboration between organizational functions and re-planning based on feedback from past iterations. Product development experts have suggested similar approaches to new product development in general. However, the adaptation rate has been faster in the software domain. This work studied knowledge transfer from Agile software development to product development involving other engineering disciplines. The research consisted of two parts, a case study and systematic literature review.

The first part of the research was a case study in an industrial setting. The case project involved teams developing electronics and mechanics. The study identified that the project benefited from knowledge transfer from Agile software development. It resulted in accelerated learning, improved communication and higher commitment. The study also identified several challenges remaining in knowledge transfer, such as larger organizational change, documentation level and the need to adapt to engineering practices.

As a second part of the research, a systematic literature review was conducted to find out what is currently known about the subject. The review found three common themes: co-design, testing and incremental hardware development. Extended collaboration between engineering disciplines was seen as important, but iterative development relying on experimentation needs new techniques for continuous testing. Despite the challenges, iterative hardware development is seen to be beneficial to system development projects. The synthesis part of the review identified an enforcing cycle between the three themes, resulting in diminishing difference between engineering disciplines.

Based on the results, it is recommended to transfer knowledge from Agile software development to new product development in general. The knowledge available accelerates the adaptation rate to a more flexible approach to development. Furthermore, it was identified that the change in product development triggers a need for change in other functions of the organization, leading toward an Agile organization. The results from the studies were mapped to a conceptual framework on how an Agile organization works. During the study, several future research avenues were identified, such as how Agile Development affects the whole organization, the economics of frequent prototyping and engineering practices, particularly regarding test automation and integration with quality assurance approaches and processes such as CMMI and ISO 9001.

Research field Computer Science and Engineering	**Key words** Agile, Scrum, new product development, hardware development
Supervising professor Casper Lassenius	**Pages** 102
Thesis advisor Casper Lassenius	**Language** English
Thesis examiner Tommi Mikkonen	**Date** 11.12.2015

☒ The thesis can be read at https://aaltodoc.aalto.fi/handle/123456789/27

Aalto-yliopisto
Perustieteiden korkeakoulu

Lisensiaatintutkimuksen tiivistelmä

Tekijä

Timo Punkka

Lisensiaatintutkimuksen nimi

Flexible New Product Development: Using Knowledge Transfer from Agile Software Development as a Catalyst for Adaptation – Case Study and Systematic Literature Review

Tiivistelmä

Tuotteiden elinkaari lyhenee jatkuvasti ja tuotekehityksen haasteisiin kuuluu jatkuva muutos nyt enemmän kuin koskaan aiemmin. Vastauksena tähän haasteeseen ketterä ohjelmistokehitys yleistyy ohjelmistoalalla. Ketterä ohjelmistokehitys pohjautuu iteraatioihin, yhteistyöhön organisaation eri funktioiden välillä ja jatkuvaan palautteen perusteella tehtävään suunnitteluun. Vastaavien menetelmien yleistyminen ohjelmistokehityksen ulkopuolella on ollut huomattavasti hitaampaa. Tässä työssä tutkittiin tiedon siirtämistä ketterästä ohjelmistokehityksestä tuotekehitykseen joka vaatii panosta myös muilta insinöörialoilta. Tutkimus koostui kahdesta osasta; case–tutkimuksesta ja systemaattisesta kirjallisuuskatsauksesta.

Tutkimuksen ensimmäinen osa oli case -tutkimus teollisuusympäristössä. Kohdeprojekti sisälsi elektroniikan ja mekaniikan kehitystä. Tutkimus paljasti, että projekti hyötyi tiedon siirrosta Ketterästä ohjelmistokehityksestä. Tuloksena oli nopeampi oppiminen, parantunut kommunikointi ja vahvempi sitoutuminen. Lisäksi tutkimus paljasti haasteita tiedon siirtämisessä. Esimerkkejä ovat koko organisaation laajuinen muutos, dokumentaation määrä ja rooli sekä tarve uusille suunnittelukäytännöille.

Tutkimuksen toisen osan muodosti systemaattinen kirjallisuuskatsaus. Tavoitteena oli selvittää mitä aiheesta tiedetään aiemman tutkimuksen perusteella. Katsaus löysi kolme yhteistä teemaa olemassa olevassa kirjallisuudessa; co-design, testaus ja inkrementaalinen hardware –kehitys. Laajempi yhteistyö eri insinöörialojen välillä nähdään tärkeänä, mutta iteratiivinen kehitys vaatii uusia käytäntöjä ja tekniikoita jatkuvaan testaamiseen. Haasteista huolimatta, iteratiivinen hardware –kehitys nähdään hyödyllisenä systeemikehitysprojekteissa. Katsauksen synteesivaihe tunnisti vahvistavan syklin kolmen teeman välillä, ja tämä johti eri insinöörialojen välisten erojen vähentymiseen.

Tutkimuksen tulosten perusteella voidaan sanoa, että tuotekehitys yleisesti hyötyy tiedonsiirrosta Ketteristä ohjelmistokehitysmenetelmistä. Saatavilla oleva tieto nopeuttaa joustavampien kehitystapojen yleistymistä. Lisäksi havaittiin, että muutos tuotekehityksessä aiheuttaa muutostarpeita myös muissa organisaation osissa. Tämä kokonaisvaltaisempi muutos johtaa ketterään organisaatioon. Molempien tutkimusten tulokset sijoitettiin konseptuaaliseen kehysmalliin, joka kuvastaa kuinka ketterä organisaatio toimii. Tutkimuksen aikana tunnistettiin useita mielenkiintoisia jatkotutkimusaiheita, kuten kuinka ketterä kehitys vaikuttaa koko organisaatioon, ekonominen ajattelu prototyyppien käytössä, kehityskäytäntöjen kehittäminen erityisesti testaukseen liittyen ja toimintatavan yhdistäminen laadunvalvonta prosesseihin ja menetelmiin, kuten CMMI ja ISO 9001.

Tutkimusala Ohjelmistoliiketoiminta- ja tuotanto	**Avainsanat** Ketterät menetelmät, Scrum, tuotekehitys, laitteistokehitys, elektroniikkasuunnittelu, mekaniikkasuunnittelu
Vastuuprofessori Casper Lassenius	**Sivumäärä** 102
Ohjaaja Casper Lassenius	**Kieli** Englanti
Työn tarkastaja Tommi Mikkonen	**Päiväys** 11.12.2015

☒ Luettavissa verkossa osoitteessa https://aaltodoc.aalto.fi/handle/123456789/27

Acknowledgements

Wow. I cannot believe it is more than ten years since I was introduced to Agile methods. I can still remember the visiting lecturer at the university pulling out slides about something called “Agile.” It was that moment that I felt, “this is how development should always have been done” -coming from a person who had had high hopes on detailed processes and detailed up-front requirement specifications signed with blood. They had all failed me miserably and I was losing hope. Agile methods were like a breath of fresh air.

I want to thank Mikko Kaijärvi, my former colleague, for being the visionary who saw the applicability of Agile methods to hardware development from my early writings on the applicability of Agile development to firmware development. He trusted the idea enough to involve me in his endeavors and gave me an opportunity to gain a deeper understanding of Agile methods on a fast track.

Special gratitude belongs to the whole product development/research ecosystem during the case study. People around the world welcomed the novel ideas and welcomed me as part of this experience. You are too many to mention, but a special thank you to Scrum Masters, Stanley Sitao-Ma and Ernesto Gomez. Cheers, Stanley and Kim: I hope our paths cross again one day.

A special thank you to James Grenning is in order. James, you have been in the role of Wise Owl for me for nearly a decade now. Starting early in the millennium as the author of the first Embedded Agile papers I was able to find, you have become a mentor and a great friend. We have experienced great adventures together. I hope there remains plenty more to be Done.

I really appreciate the many hours that Kristian Rautiainen put into guiding me while struggling to get the paper to at least resemble an academic style and quality. I was amazed by your expertise on the practical side and capability of understanding how a person from industry has trouble presenting ideas with the precision that academia requires. I truly believe that the paper reached a whole different level with your guidance.

Professor Casper Lassenius, like you once said, the delivery of the thesis got delayed by a “few” weeks. Thank you for believing in this project despite this and not – totally – giving up on me!

Thanks to my many reviewers, Nancy Van Schooenderwoert, James Grenning, Rolf Østergaard, Niel Johnsson and Tobias Leisgang. Thanks also to you as a group on behalf of the Agile system engineering community for your contribution to the body of knowledge on knowledge transfer. It’s been a great

honor to be accepted as an equal practitioner (at least I have felt it has been like that). Keep up the great work!

Last, but definitely not least, I want to thank my family, my wife Pia and my son Nico, for believing that I could finally get the thesis completed. I know it took a lot of my time away from you, and I truly appreciate you giving me the chance to accomplish this. I hope from now on you will hear a bit less of, “I’ll be there in a sec” from behind the computer from me.

Finns are known for being people of few words. On that note, I close this part of my journey.

Helsinki, December 2015
Timo Punkka

Contents

ACKNOWLEDGEMENTS **1**

CONTENTS **3**

1 INTRODUCTION **5**

1.1 Motivation 5
1.2 Objectives and scope of the research 8
1.3 Research approach 8
1.4 Terminology 9
1.5 The structure of the thesis 10

2 BACKGROUND **11**

2.1 Agile software development 11
2.1.1 Scrum *15*
2.1.2 Extreme Programming *16*
2.2 Flexible Product Development 17
2.2.1 Motivation *19*
2.2.2 Whole team approach *19*
2.2.3 Emergent process and design *20*
2.2.4 Change beyond engineering *21*
2.3 Knowledge transfer from software domain to hardware domain 22

3 RESEARCH DESIGN – CASE STUDY **25**

3.1 Research approach 25
3.2 Case project 27
3.3 Research process and data gathering 29
3.4 Data analysis 31
3.5 Threats to validity 32

4 RESULTS – CASE STUDY **34**

4.1 Case project 34
4.1.1 The project's front-end *34*
4.1.2 First part of project: Proof of concept *36*
4.1.3 Middle part of project: Validation of the architecture *39*
4.1.4 Last part of project: Preparation for production *43*
4.2 Key findings 45
4.2.1 Accelerated learning through up-front prototyping *46*
4.2.2 Improved communication *53*
4.2.3 Improved commitment *59*
4.2.4 Remaining and new challenges *63*

5 RESEARCH DESIGN – SYSTEMATIC LITERATURE REVIEW **70**

5.1 Research method 70
5.2 Database search strategy 71
5.3 Primary study selection 75

5.4 Creating a secondary study 76

6 RESULTS – SYSTEMATIC LITERATURE REVIEW 78

6.1 Overview 78
6.2 Key findings 80
6.2.1 Co-design 80
6.2.2 Testing 82
6.2.3 Iterative hardware development 83
6.2.4 Secondary interpretation: Enforcing cycle 85

7 DISCUSSION 87

7.1 Summary of the case and literature studies 87
7.2 Comparison of the case and literature studies 90
7.3 Further discussion on organizational implications 92
7.4 Answering the research questions 95
7.4.1 Research question 1 95
7.4.2 Research question 2 96
7.4.3 Research question 3 96
7.5 Limitations 97

8 CONCLUSIONS 99

8.1 Summary and conclusions 99
8.2 Future research 100

9 REFERENCES 103

APPENDIX A – DATA EXTRACTION FROM PRIMARY STUDIES 109

APPENDIX B - INTERVIEW INSTRUMENT A 118

APPENDIX C - INTERVIEW INSTRUMENT B 119

1 Introduction

The opening chapter provides the overall motivation for the research and thesis. It starts by illuminating the need to study the process of knowledge transfer from the software industry to new product development in general. The subsequent sub-sections present the research problem, specific research questions, the research approach and the terminology used in the thesis. Finally, the chapter introduces an overview of the structure of the whole thesis.

1.1 Motivation

Increasing uncertainty is one of the most daunting challenges organizations and industries are facing today. Doz and Kosonen (2008) list destabilization forces that have strongly changed the rules of the game for organizations battling for revenue:

- Digitalization
- Globalization
- Deregulation

The uncertainty is present in multiple dimensions, such as technology, competition and the market place. The change has led to a situation where even industry boundaries are in flux (Hamel and Prahalad, 1994). In short, uncertainty is increasing. Growing uncertainty affects all companies, not least their new product development. Product development project management practices have traditionally expected a linear execution from requirement gathering through implementation to testing as final verification. The simplicity of this linear model is defeated by the growing uncertainty and accelerating rate of change that comes with it. It becomes clear that instead of developing processes which try to avoid change, we need to develop processes which cope with uncertainty. As early as 1994, Ogunnaike and Ray distinguished these two approaches to process development as *defined* as compared to *empirical* approaches:

> *"It is typical to adopt the defined (theoretical) modeling approach when the underlying mechanisms by which a process operates are reasonably well understood.*
>
> *When the process is too complicated for the defined approach, the empirical approach is the appropriate choice."*

The rest of the thesis will reference the empirical approach as flexible product development (Smith, 2007). The need for flexible product development may

be stronger today than ever before, but it is interesting that empirical, iterative models have been promoted for a long time.

The term “Waterfall model” is used to describe a defined and sequential approach to product development. In this model, the project is seen as progressing through phases. A project only advances to the next phase after fully completing the previous phase. Winston Royce’s paper “Managing the Development of Large Software Systems” (Royce, 1970) is often referenced as the source of the Waterfall model. This reference is at least questionable, because Royce actually argued that this naïve approach fails in anything but the most simplistic projects. Nevertheless, in his original paper, the phases follow each other in order: requirement gathering, design, construction, integration, testing and debugging, installation and maintenance.

Other sources that are claimed to “demand sequential processes” are, in reality, not that restrictive, either. The Project Management Institute (PMI) lists the sequential model in their Project Management Body of Knowledge (PMBOK) as one of the possible project life cycle models. As equally considerable alternatives, the guide presents models with overlapping phases and iterative approaches.

The Stage-Gate process model was first introduced in 1986, when the first edition of “Winning at New Products” was published by product development expert Dr. Robert Cooper (Cooper, 1986). The name for the model comes from the fact that it promotes dividing projects into stages. Stages are separated by screening activity to make a go/kill decision; in other words, gates. The stages were called scoping, building the business case, development, testing & validation and launch. Like Winston Royce and PMI, Cooper also recommends more iterative approaches. He does not, for example, see development as an isolated effort, despite the fact that it has a separate stage:

> *“… parallel marketing and operations activities are also undertaken. For example, market-analysis and customer-feedback work continue concurrently with the technical development, with customer opinion sought on the product as it takes shape during development. These activities are back-and-forth or iterative, with each development result – for example, rapid prototype, working model, or first prototype – taken to the customer for assessment and feedback."*

Despite the above recommendations, new product development in general (including development of physical products) has been slow in adopting flexible product development. Cooper (2009) references studies conducted by the Product Development and Management Association (PDMA) and the American Productivity & Quality Center, and concludes that 70 percent of product developers in North America use the State-Gate process model or

similar. The referenced studies were conducted in 2004 and 2002 respectively. In 2010, PMI's Pulse of Profession white paper (PMI, 2010) stated that 39% of respondents used a Waterfall model (survey of over 1,100). Based on this, it is possible to conclude that the phased and sequential process model dominates new product development in industry.

When we consider the recent history of software industry, we find it to be significantly different in terms of process development compared to new product development in general. The growing uncertainty has of course also affected the software industry. The history of iterative and incremental software development models is decades old, but public awareness increased in the late 1990s (Larman and Basili, 2003). In 2001, the common name "Agile software development" was coined to mean models that share similar values and principles (Highsmith, 2001). Agile methods focus on exploration and collaborative learning between business and development throughout product creation. Incremental and iterative development is steered by continuous re-planning based on feedback. Changes in requirements are welcome and seen as an opportunity rather than a threat. Compared to trends in new product development in general, Agile software development has become rapidly popular. In 2010, the Forrester report concluded that based on their survey of 1,300 IT professionals, 35 percent are using some variant of Agile software development (West and Grant, 2010). The percentage increases if we include responses for iterative (16%), Rational Unified Process (3%) and Spiral (2%) models. One-third of respondents used no formal methodology and only 13 percent claimed to use the Waterfall model. The results achieved with Agile software development are also encouraging. The Agile Impact Report by QSM Associates in 2008 concluded that there were on average improvements in three main areas: a 37% faster time-to-market, 16% more productive and no rise in defect count despite the compressed schedule (QSMA, 2008). The study benchmarked 29 projects using Agile development methods against 7,500 primarily traditionally managed projects.

To sum up, flexible product development has been recommended for a long time. Concrete guidance also exists (Smith and Reinertsen, 1997; Reinertsen, 1997). Nevertheless, Agile software development has been significantly more popular in the industry than approaches developed for new product development in general. In systems development, the problems that Agile software development aims at helping are shared between multiple engineering disciplines. Because of this, it is tempting to consider whether there is something that industry as a whole can learn from Agile software development and the software domain. Can the knowledge created in the software domain be used to accelerate process development in other engineering disciplines or product development in general? This is especially interesting because of the trend of growing software intensity in the products that companies develop (Boehm, 2006). This reasoning is further explored in chapter 2.3.

1.2 Objectives and scope of the research

The main objective of this thesis is to find out if the knowledge and ideas behind Agile software development can be transferred to projects requiring work from other engineering disciplines - helping the adaptation of flexible product development. Therefore, the research problem is:

> *Can the knowledge from Agile software development be transferred to supplement flexible product development processes and accelerate their adoption in other engineering disciplines and system development?*

The research problem can be divided into three research questions:

> *RQ1. Can knowledge from Agile software development be transferred to non-software new product development?*
>
> *RQ2. What are the implications of introducing concepts familiar from Agile software development in the development of physical products?*
>
> *RQ3. Does Agile development impose larger organizational implications?*

In this study, we are only interested in knowledge transfer from Agile (as in the software development domain) and its implications for non-software new product development. The fact that the case study took place in a distributed environment obviously affected the case, but distribution itself is outside the scope of this study. The case study is introduced in detail in chapter 3.2.

1.3 Research approach

In order to answer the research questions, a two-part study was conducted. The first part was a case study of hardware development using knowledge from Agile software development. The case project was in a globally distributed industrial setting. The project lead believed that the knowledge he had gained from Agile software development would help in the project setting, although the project was focused on the hardware domain. Because this was a project in an industrial setting, there was no time for thorough research planning, but it was decided that data should be collected for research purposes. During the study, the author helped the project organization in the knowledge transfer from Agile software development. After the case study, a systematic literature review was conducted to learn what is already known about knowledge transfer. The systematic literature study formed the second part of the research for the thesis. The findings from the two parts were used to explore the main research problem by analyzing the degree to which the existing research supports the observations from this particular case. The limitations of

conducting the two parts in this order, and not in the conventional order of conducting the literature study first, are discussed in chapter 7.5. In summary, the approach is justified simply because an opportunity for experimentation was offered in an industrial setting. A more detailed research design for the case study is presented in chapter 3 and for the literature study in chapter 5.

1.4 Terminology

This sub-chapter defines the terminology used in the thesis. The objective of the list is not to be inclusive, but to provide sufficient understanding for the reader to follow the text. For this reason, this section does not cover terms that are adequately familiarized in the context.

Capability Maturity Model Integration (CMMI) is a process maturity and improvement framework. It models organizational improvement using five maturity levels, each containing a set of requirements for the organization.

Co-design stands for the parallel development of software and non-software components of a system.

Cross-disciplined means different engineering disciplines working together toward a shared goal.

Cross-functional means representatives from different organizational functions (design, testing, manufacturing, marketing, etc.) working together toward a shared goal.

Distributed development means development where the effort is divided between two or more different sites, but still sharing the same goal.

Hardware development means the development of physical elements of products, such as electronics, mechanics, industrial design or design for manufacturing.

Incremental development means adding something to the design, for example, a new feature.

ISO9001 is a standard developed by the International Organization for Standardization (ISO), which defines how an organization defines, executes and maintains a quality assurance system. Independent certification bodies provide assessments for organizations looking to become certified.

Iterative development means refining, or reworking, a design.

Iteration means a time-boxed (fixed time slot) activity delivering a coherent set of value.

New Product Development (NPD) means any kind of development. Development can be just software or can include hardware or any other non-software elements as well.

Non-software means any other development goal than software, and can include for example hardware or services development.

Project Management Body of Knowledge (PMBOK) is a collection of project management knowledge maintained by the Project Management Institute.

Project Management Institute (PMI) is a non-profit organization concentrating on knowledge creation for project management professionals.

1.5 The structure of the thesis

The thesis is structured into eight chapters: Introduction, Background, Research design – case study, Results – case study, Research design – systematic literature review, Results – systematic literature review, Discussion and Conclusion. Chapter 2 gives deeper background information for the research. Chapter 3 presents the chosen inquiry strategy for the case study. Chapter 4 presents the conclusion of the case study. Chapter 5 presents the framework for conducting the systematic literature review. Chapter 6 contains the results of the systematic literature review. Chapter 7 summarizes the individual studies and discusses the two studies together. The latter part provides answers to the research questions and considers the limitations of the research. Finally, chapter 8 provides a summary and contribution of the research as a whole. The last sub-chapter lists identified areas for future research.

2 Background

This chapter provides background information for a deeper understanding of the research problem. The background begins with an overview of Agile software development and continues with a specific introduction to two Agile methods, Scrum and Extreme Programming. Following Agile software development, the next sub-section reviews flexible product development in other fields. The final sub-section considers Agile software development and flexible product development together and examines the possibilities and reasons for knowledge transfer between the two domains.

2.1 Agile software development

In the 1990s, several different lightweight methods were developed with the goal of making software development more successful by adapting it better to continuous change, which seemed inevitable. The common use of the term Agile started after the creation of the Agile Manifesto in 2001 (Highsmith, 2001). The cover page of the manifesto states the values of Agile software development as follows:

> *We are uncovering better ways of developing software by doing it and helping others do it. Through this work, we have come to value:*
>
> *Individuals and interactions over processes and tools*
>
> *Working software over comprehensive documentation*
>
> *Customer collaboration over contract negotiation*
>
> *Responding to change over following the plan*
>
> *That is, while there is value in the items on the right, we value the items on the left more.*

The Agile Manifesto was the result of 17 software method developers meeting to discuss their ideas on software development to see what they all have in common. The manifesto summarizes the shared understanding of better software development. Beyond the values stated above, the Agile Manifesto lists twelve principles, explained in Table 1.

Table 1. Agile principles (Highsmith, 2001; Cockburn, 2001).

Agile Manifesto Principle	**Explanation**
Satisfy customer through early and frequent delivery of valuable software	You can't ship specifications or design models if the customer has ordered product.
Welcome changing requirements even late in the project	Rather than resist change, the Agile approach strives to accommodate it as easily and efficiently as possible.
Keep delivery cycles short (e.g., every couple of weeks)	Each delivery should have some additional value to customer, but deliver is not necessarily the same as release.
Business people and developers work together daily throughout the project	High level view of requirements is not enough for development, so the gap is closed with frequent interaction between the business people and the developers.
Build projects around motivated individuals	People are the ultimate success factor for a project. People knowing the most about the situation must be the ones making the decisions.
Place emphasis on face-to-face communication	Usually the problem is not the lack of documentation, but the lack of understanding. Face-to-face communication reduces the chances of misconceptions.
Working software is the primary measure of progress	The actual product provides milestones and accurate measures of progress. By delivering often the details of requirements can be captured in small steps.
Promote sustainable development pace	Back in the days it was glamorous to put in long nights and weekends, but Agile methods need alert people and those long nights do not actually provide greater productivity anyway.
Continuous attention to technical excellence and good design	Quality should be an integral part of development. Design issues get more costly to handle over time.
Simplicity – the art of maximizing the amount of work not done - is essential	In an Agile project, it's particularly important to use simple approaches, because they're easier to change. It is easier to add something to something simple, than to take away something from something that is complex.
The best results emerge from self-organizing teams	The best architecture, requirements and design emerge from teams in which interactions are high and the process rules are few.
Team reflects regularly where and how to improve	An Agile team continuously refines its process and methods to improve and to match the changing circumstances.

Agile development acknowledges that it is impossible to plan the project completely in detail at the outset. The traditional model is based on the opposite assumption that all information about the project is available in the beginning. If this was true, then it would be most effective to plan the project carefully and then execute accordingly. However, development is typically not certain about how the technology works. Similarly, the business learns about the markets continuously. New information arrives steadily throughout the project, creating a need for change. Agile development addresses this change by relying on people from multiple functions working together in short iterations and gathering new information in terms of feedback from short experiments. Experiments are targeted toward a functioning product, bringing testing activities forward thus further increasing the feedback bandwidth. The concept is only closed for change, frozen, when enough information has been

gathered. In other words, the phases familiar to the traditional process model are executed in parallel rather than sequentially, as illustrated in Figure 1.

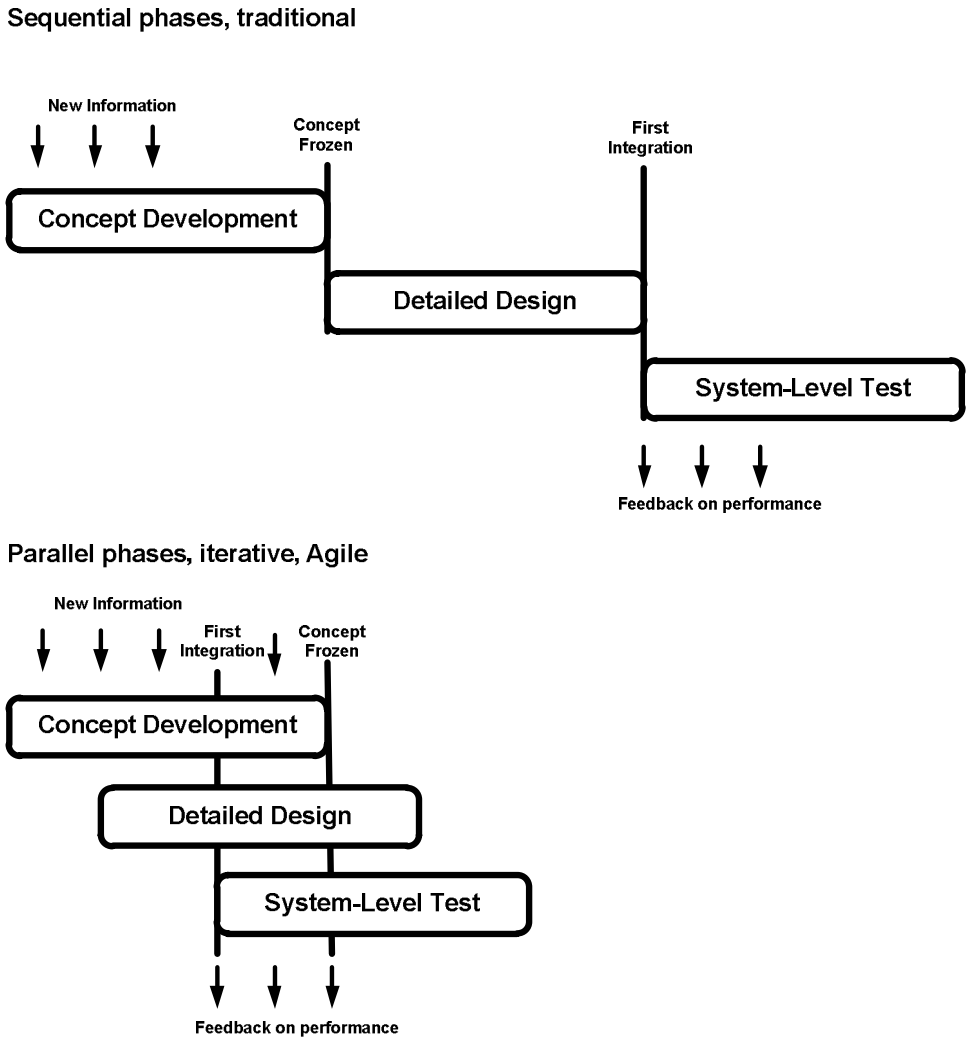

Figure 1. Agile development creates information using short iterations and feedback on the functioning product (MacCormack, Verganti and Iansiti, 2001).

As already stated, the parallel model implies that learning and knowledge creation is required by everyone, including business and development. While the business learns about the changing markets, the development learns from technical feasibility. The new discoveries on either side further trigger the need to experiment and refine the solution. The interesting characteristic of disciplined Agile development is the simultaneous learning on the business side (in terms of product features and capability) and technical solutions using fast-paced, time-boxed experiments or iterations (Figure 2).

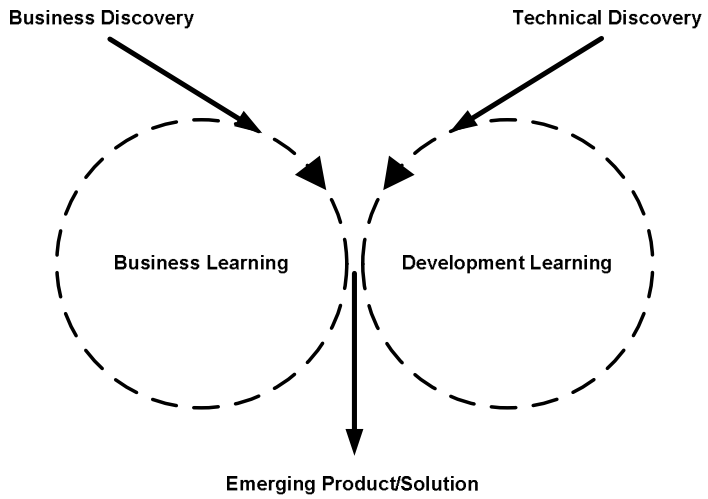

Figure 2. Collaborative learning between the business and the development .

Although the individual Agile practices are not new (Cockburn and Highsmith, 2001), the underlying philosophy is very different from traditional thinking in many development organizations. Table 2 summarizes the differences between traditional and Agile software development (Nerur, Mahapatra and Mangalaraj, 2005).

Table 2. Traditional versus Agile software development (Nerur, Mahapatra and Mangalaraj, 2005).

	Traditional	**Agile**
Fundamental assumptions	Systems are fully specifiable, predictable, and can be built through meticulous and extensive planning	High-quality, adaptive software can be developed by small teams using the principles of continuous design improvement and testing based on rapid feedback and change
Control	Process centric	People centric
Management Style	Command-and-control	Leadership and collaboration
Knowledge Management	Explicit	Tacit
Role Assignment	Individual – favors specialization	Self-organizing teams – encourages role interchangeability
Communication	Formal	Informal
Customer's Role	Important	Critical
Project Cycle	Guided by tasks or activities	Guided by product features
Development Model	Life cycle model (Waterfall, Spiral, or some variation)	The evolutionary-delivery model
Desired Organizational Form/Structure	Mechanistic (bureaucratic with high formalization)	Organic (flexible and participative encouraging cooperative social action)
Technology	No restriction	Favors object oriented technology

Today, Agile software development is a term used for a wide range of incremental and iterative development methods, methodologies and frameworks. Examples of Agile software development models are Scrum, Extreme Programming (XP), Feature-Driven Development (FDD), DSDM and Crystal Family. Out of these, Scrum is the mostly adopted in industry. The training and tool company VersionOne has been conducting a survey on the Annual State of Agile Development for many years. The 7th annual survey in 2013 revealed that 54% of practitioners are using Scrum (Versionone, 2013). The significance of Scrum is even more visible, since an additional 11% reported using a combination of Scrum and Extreme Programming. The next sub-sections introduce Scrum and Extreme Programming in more detail.

2.1.1 Scrum

Scrum (Schwaber and Beedle, 2002; Schwaber, 2004) is a project management framework illustrated in Figure 3. In Scrum, work is broken down to fit short iterations called Sprints. The work to be done in a given Sprint is planned in a Sprint Planning Meeting. In this meeting, work is pulled from a Product Backlog, and moved into the team's Sprint Backlog. The Product Owner is responsible for prioritizing the Product Backlog containing all the work to be done in a project. However, the work planned for a given Sprint is managed by the self-organizing Scrum team. A Scrum team consists of 10 or fewer members. Ideally, the team has all the skills necessary to complete the Sprint. During the Sprint, the team has a Daily Scrum Meeting to synchronize the information. The Scrum Master is responsible for keeping the process fit and coaching the team in continuous improvement. At the end of each Sprint, the Scrum team demonstrates its achievements to all stakeholders in a Sprint Review meeting. Sprint Review serves two primary purposes. It demonstrates the team's progress in a concrete way, and it offers an opportunity to give and receive feedback. Between Sprints, the team holds a Retrospective meeting to gather improvement ideas. Scrum does not give guidance on engineering practices. For this reason, teams often supplement it with practices from other methods, such as Extreme Programming (Beck, 2000; Beck, 2004). Schwaber and Beedle (2002) express this thus: "If practices were candy the Scrum is the wrapping paper for candy." This fact also makes Scrum interesting from the perspective of other engineering practices. There is very little, if anything, in Scrum that is unique to programming. Instead Scrum gives guidelines for incremental and iterative planning in general. This guidance can be considered for any kind of work.

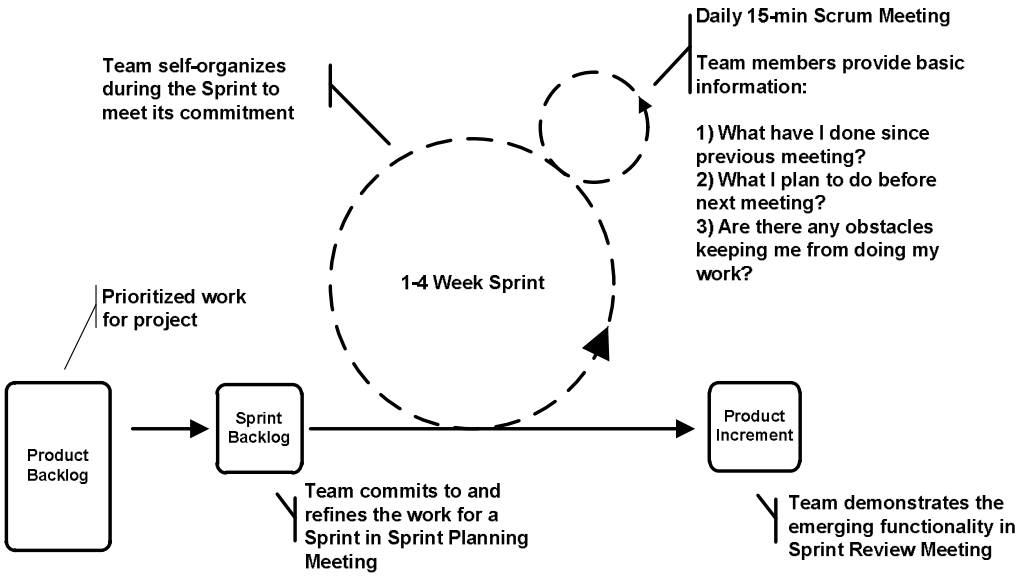

Figure 3. Scrum Framework (Schwaber and Beedle, 2002; Schwaber, 2004).

2.1.2 Extreme Programming

While Scrum provides little guidance for actual programming, Extreme Programming (XP) is very prescriptive on programming practices and techniques. Extreme Programming was defined in a real project, and was introduced to wider audience by Kent Beck (2000; second edition, Beck and Andres, 2004). XP has five values forming the boundary: communication, simplicity, feedback, courage and respect (the last was added in the second edition).

Building on those values, XP provides a list of principles which define the overall philosophy of the method: humanity, economics, mutual benefit, self-similarity, improvement, diversity, reflection, flow, opportunity, redundancy, failure, quality, baby steps and accepted responsibility.

The first edition (Beck, 2000) presented 12 practices that describe how to *do* Extreme Programming. The practices can be thought as known good practices, but executed at the extreme level. The original 12 practices are introduced in Table 3, and some are referenced elsewhere in the thesis. The second edition (Beck and Andres, 2004) defines even more practices. The first 13 are primary practices, which are generally similar to the original 12 practices presented below. The second edition further introduces corollary practices, which differ in that they need support from other practices.

Table 3. Original twelve practices in Extreme Programming (Beck, 2004).

Practice	Description
Planning	Planning is collaborative activity between development and business. Business provides the priority, while development provides the cost estimate.
Small Releases	Product is launched iteratively in order to enable feedback as soon and frequently as possible.
Metaphor	The business and development need to be able to collaborate about the product using a language they both understand.
Simple Design	The design of the system is only what is needed for the current functionality.
Testing	Automated testing is happening on many levels, such as unit tests and acceptance tests.
Refactoring	Code is under continuous refinement for better design.
Pair Programming	Two programmers working on the same computer is an extreme way of peer reviewing and continuous collaboration on design.
Collective Ownership	Anyone on the project can work on any part of the system.
Continuous Integration	Each change in the source code triggers an automated build process, including automated test suites.
40-hour Week	Overtime is avoided in order to achieve sustainable pace.
On-site Customer	A customer, or a representative, is working all the time with the developers on the team.
Coding standard	When the team is working on the same source code, it is important to follow the same coding conventions improving communication.

Although Extreme Programming is about exactly what the name suggests, programming, it does not mean that we should overlook these practices in hardware development domain. If you again take a look at the table above, you can find that most of the practices are in fact applicable to other engineering projects. Practices regarding iterative planning, continuous testing, and working together collaboratively with other people are worth consideration in any environment. We will see this in practice later in the thesis.

2.2 Flexible Product Development

Smith (2007) analyzed how teams developing other products than software can use similar practices to those used in Agile software development. He called this approach flexible product development, as an alternative to traditional, linear, process models. Smith argued against using the term Agile in this context. He felt that the Agile principles and practices from software development do not directly translate to product development in general, but the approach needs to be rebuilt. This argument resonates very well with the objective of this study. However, we do not need to start this rebuilding from scratch. We can build on knowledge created in different domains. In chapter 5, we already concluded that general recommendations for flexible alternatives to

the new product development process exist in the literature. In order to gain a deeper understanding of what is already known and actually recommended, we took a wider look at articles addressing flexible new product development. The articles for the review were selected based on prior knowledge. Each of the references was read through and summarized in writing. Similarities from summaries were grouped into themes. Four themes emerged: motivation, whole team approach, emerging process and design, and change beyond engineering.

Table 4. Summary of articles in chronological order.

Reference	**Summary**
Takeuchi and Nonaka, 1986	Whole team • Heavily relying on self-organizing teams Emergent process and design • Development acting as a source of change for the whole organization Change beyond engineering • Management executes only subtle control • Holistic approach is company-wide-learning-driven
Smith, 1990	Motivation • Presents methods to justify increased cost, and reduced scope, to get products to production faster Whole team • Calls for continuous collaboration across the organization's functional boundaries Emergent process and design • Promotes iterative model building Change beyond engineering • Even senior management works with development from the outset
Thomke and Reinertson, 1998	Motivation • Identifies two major reasons for need for flexibility; products are more complex and markets are more volatile Emergent process and design • Flexible design to tolerate the changing market needs
Clay and Smith, 2000	Motivation • Iterative prototyping accelerates development, resulting in higher quality Emergent process and design • Prototypes to be used for risk management and as measure of progress resulting in faster time-to-market • Concrete guidance on using prototypes for focused learning
Cooper and Edgett, 2009	Motivation • Reasons for development failures, lack of user input, unstable specifications and missing real teams Whole team • Holistic approach using cross-functional teams Emergent process and design • Focusing on customer and using iterations Change beyond engineering • Focused portfolio management using the funneling approach

Table 4 summarizes the findings from the literature. The themes are well aligned with Agile software development. Especially interesting is having the development team in a central role, and acknowledging the need for emergence in both the product being developed and the process itself. Agile software development and flexible product development are further discussed together in the last sub-section.

2.2.1 Motivation

Several authors state reasons why companies should move toward more flexible product development, giving us the *motivation* to explore novel approaches to product development. In his paper, "Fast-Cycle Product Development," Preston Smith (1990) makes the point of the increasing need for development speed. He presents a model for the economics of development speed, proposing a simple model to justify the development cost if it gets the product to the market earlier, thus enabling cash flow sooner. Thomke and Reinertson (1998) take a slightly different angle, but they call for development speed as well. They identify two reasons for the increasing need of flexibility in development. First, products are becoming more complex. Because devices have more functionality, the task of specifying this functionality becomes more difficult. Furthermore, they state that the rate of change in most markets is increasing. The most obvious weapon to tackle the challenge of growing complexity and increasing rate of change is reducing the development time. This means that the time during which the product is vulnerable to change shortens. Short increments using rapid iterative prototyping are also recommended by Clay and Smith (2000). This is considered to be a method of proactive risk management and to greatly accelerate product development and lead to high, defect-free quality. Along similar lines are Cooper and Edgett (2009), who propose a more flexible updated version of the Stage-Gate model as a solution for product development failures. Reasons to these failures that they list include lack of user input, unstable product specifications and no real project team.

2.2.2 Whole team approach

Authors recommend a *whole team approach* utilizing cross-disciplined and cross-functional teams. In the classic paper, "The New New Product Development" published in Harvard Business Review, Hirotaka Takeuchi and Ikujiro Nonaka (1986) present their findings from six successful product development projects. As early as 1986, this paper presented ideas that may sound novel to many product development organizations today. The projects they studied developed copiers, cameras and a car. What they observed was completely different from many less successful projects. They call this a holistic approach. At the core of this approach is the self-organizing team. The

team is truly cross-functional and consists, for example, of members from development, production and sales. The team must have the autonomy to decide its way of working. It must exhibit self-transcendence, the ability to raise the bar and because of its cross-functional structure, has self-fertilization, an ability to share and create knowledge. In the same way, several of the suggestions by Smith (1990) to help organizations get faster focus on a whole team approach with strong teams. A strong team has strong and creative leadership. Members are full-time, to increase commitment and accountability. The team is small and cross-functional, which fosters continual direct communication. It enables frequent synchronization of partial information, and, for example, joint specification. Keeping the communication informal is encouraged to further speed things up compared to other communication formats, such as written plans, specifications, reports and reviews. Cooper and Edget (2009) have a similar message with their practice of a holistic approach driven by effective cross-functional teams.

2.2.3 Emergent process and design

Emergent process and design is another common theme in the flexible product development literature. Clay and Smith (2000) emphasize that the biggest gains can be achieved at the beginning of the project. Quick experiments with rapid prototyping techniques lead to consensus about the optimal design, greatly shortening the overall design time. Every prototype should be aimed at answering specific, quite narrow questions, and they should be designed only at a level sufficient (detail, robustness, etc.) to answer the question. Multiple competing solutions should be prototyped in parallel. The answers from the current prototype are transferred as knowledge and decisions to the next prototype round. In general, the faster the prototyping rounds, and therefore learning, the faster the product development cycle. Clay and Smith also report organizations using working prototypes as a project management tool. Prototypes can be used as a vehicle for communication between marketing, management, focus groups, etc. The prototype can be seen as an emerging product, a living specification, as well as a progress report. An updated Stage-Gate model guides us in the same direction (Cooper and Edget, 2009). The Stage-Gate model has evolved to incorporate more flexibility and acknowledge fluid information and the need to experiment iteratively. Inside a stage, the project progresses through a series of “build-test-feedback-and-revisit” iterations. They call this spiral development. Smith (1990) recommends model building with “cut-and-try” cycles. Thomke and Reinertson (1998) have a slightly different angle. They make the point that the design itself can also be suitable for late changes, especially in areas where we can foresee a possible need for a change. They propose the following definition for flexibility in the product development context:

> *"Development flexibility can be expressed as a function of the incremental economic cost of modifying a product as a response to changes that are external (e.g. a change in customer needs) or internal (e.g. discovering a better technical solution) to the development process. The higher the economic cost of modifying a product is, the lower the development flexibility is."*

Emergence acknowledges that product development is most of all about learning. According to the research by Takeuchi and Nonaka (1986), successful organizations engage in "Multi-Learning." When the process happens in all functions in a more parallel than sequential way, the organization learns at different levels: individual, team and corporate. Furthermore, parallel development creates a shared rhythm for the company. People also learn from multiple functions, gaining knowledge from broader areas. Organizations intensively enforce knowledge transfer. This can happen in multiple ways, such as circulating project members and standardizing current practices.

2.2.4 Change beyond engineering

Many authors also suggest that changing product development to be more flexible entails *a change beyond engineering*. Smith (1990) calls for senior management involvement in the outset of the project, during the concepting phase. This may be new in many organizations, because it involves being comfortable with an uncertain and vague vision. Experimenting also requires tolerance of failures. Too often companies have defined procedures and controls to avoid mistakes. Protective processes and heavy governance hinder the speed of decision-making, which is required in iterative experimenting. Nonaka and Takeuchi (1986) remind us that management should execute only subtle control, managing the environment and being responsible, for example, for recruiting and educating external suppliers. This changes the traditional line management responsibilities quite dramatically. The cost of involving a large number of people, from senior management to floor-level engineering, is earned back later because the traditional review phase in the end shrinks to a mere final OK check, since everyone is already on the same page. Change beyond engineering, and working more in parallel, is needed to optimize the whole time it takes to transfer ideas to profit (Smith, 1990). Smith gives the practice of limited product objectives as an example. The functionality of the product is sacrificed (Limited Product Objectives) in order to keep the complexity manageable and shorten the time-to-market. This is quite different from the Waterfall model's practice of trying to gather all the possible requirements before beginning development. Another concrete example of larger organizational impact is focused portfolio management using the funneling approach to select targets for development investment (Cooper and Edgett, 2009).

2.3 Knowledge transfer from software domain to hardware domain

So far we have looked at Agile software development in general, specific Agile methods and literature on flexible product development. It is time to think about what we can learn from all this. First, it seems that ideas in Agile software development are not so strictly tied to the software domain. Second, guidance for flexible product development is very similar to ideas in Agile software development. In chapter 1.1 we already concluded that it might be interesting to study the knowledge transfer from the software development domain to the hardware development domain or product development in general. Now is a good time to explore the motivation for the knowledge transfer more thoroughly.

Interestingly, the Scrum framework introduced in chapter 2.1.1 was inspired by the classic paper, "The New New Product Development Game" by Takeuchi and Nonaka (1986) referenced in chapter 2.2. The paper in fact first used the term Scrum. Therefore, the Scrum framework for software development actually transferred knowledge from the field of physical product development into the software industry. The paper by Takeuchi and Nonaka did not describe software development, but product development characteristics in general. Since then, the Scrum and Agile software development communities have made a giant leap in defining the practices for software development in more detail. Agile software development learned from the field of physical product development. It makes one wonder whether it would work the other way around as well. Authors have presented this idea, mapping values, practices and even techniques from Agile software development into development in other engineering disciplines (Smith, 2007; Highsmith, 2009).

Hence, these ideas are not new, but the evidence from industry is scarce. As Smith (2007) states in his introduction chapter in "Flexible Product Development," "However, this presents a paradox for a new field with only limited experience to present in demonstrating how the techniques apply to development projects." He reminds us that if there were plenty of experience data, the idea would not be new.

Why would this be interesting now? Examples of reasons are that development challenges are shared between engineering disciplines, Agile software development is being rapidly adopted in industry and the software intensity in products that we develop is growing. These three reasons are summarized in Table 5.

Table 5. Summary of reasons for knowledge transfer.

Reason	Description
Problems are shared and similar	In system development the challenges are shared by different engineering disciplines. Agile software development aims at tackling these challenges.
Adoption rate of Agile software development	Agile software development is being rapidly adopted in industry.
Increasing software intensity	In many organizations the share of software development, and therefore Agile development, is growing, making Agile the dominating process model.

First, development in different engineering disciplines *shares similar problems*:

- Products need to get to market faster
- Increasing amount of change (or learning) during development
- Products to be developed are getting more complex

Agile software development addresses these challenges. It is being rapidly adopted, and the results reported are very positive. Highsmith, one of the original authors of the "Agile Manifesto," proposes that in systems development, the whole development team needs to be aligned with Agile development. It is not just the software discipline of product development (Highsmith, 2002).

Second, the *adoption rate of* Agile software development is much higher than flexible product development in other areas (discussed in chapter 1.1). Of course, software has characteristics that make it more suitable for rapid experimenting, but one reason might be that the amount of knowledge on the flexible development approach is higher for Agile software development. Concrete, practical, example is the concept of iteration. The flexible product development literature presents the need for it, but the practical implementation details are left for the practitioner to figure out. In contrast, the Agile software development literature is full of information on how to conduct iterative and incremental development collaboratively across the organization. In other words, the Agile software development community has defined concrete practices, which enable a quick start. On the other hand, Cockburn and Highsmith (2001) state that the practices collected under the common name of Agile methods were not new, but the novelty came from the focus on a new combination of values and principles. Agile software development provides knowledge from that perspective as well.

Third, the *software intensity* in products is increasing all the time. Companies developing products that have traditionally been conventional electrical devices are facing a new situation, as software development is taking up increasing effort in product development. Usually, the institutionalized

product development process is aimed at traditional hardware development. In many cases, this is rigid Stage-Gate implementation. As we see Agile software development moving into the mainstream, there is a conflict of mind-sets between the now-dominating software development and the rest of the organization. It helps if we are able to explain Agile software development in the context of other engineering disciplines.

All this makes it interesting to think about transferring knowledge from one profession to the other, maybe bringing them closer to each other. Ovesen (2012) presents the idea of transferring knowledge from the software domain to the integrated product development domain. His study focuses on identifying the challenges in this transfer. The challenges the study lists cannot be said to be completely caused by the fact of developing a physical product. For example, challenges faced by developers include: breakdown of work, estimating, tangible achievements in Sprint Reviews and keeping design flexible for future changes. Furthermore, when implementing Scrum in the integrated product development environment, Ovesen identifies organizational challenges such as motivating cross-functional teams, high-performing team composition, changes in management approach, and handling disturbances from the surrounding organization. Again, I do not fully agree that these challenges are unique to transferring Agile development from the software domain to other domains. Rather, I see these as quite common challenges in organizational change.

As a summary, authors have presented the idea of knowledge transfer from Agile software development to product development in general. There is an evident promise, but the experimental data is very limited. This thesis continues to explore this knowledge transfer.

3 Research design – Case study

This chapter discusses the research design for the case study. First, it presents the selected research approach and introduces the case under study. The middle part of the chapter describes the process for data gathering and data analysis. The final sub-chapter considers the threats to the validity of the research.

3.1 Research approach

The research questions were introduced in chapter 1.2. The key decision in planning an inquiry is the application, or purpose, of the research. Yin (1994) lists five different research applications; explaining, describing, illustrating, exploring, and meta-evaluation. The applications are illustrated as a pyramid in Figure 4. One can say that the higher in the pyramid the application gets, the more needs to be known about the subject of the study. In that sense, the applicability of quantitative methods seems to be higher at the top of the pyramid.

The objective of this study is to add to the body of knowledge on knowledge transfer from Agile software development to new product development in general, as called for by Preston Smith (2007). Therefore, the research application is *explorative*, as suggested for situations and phenomena in which the intervention has no clear expected outcome (Yin, 1994).

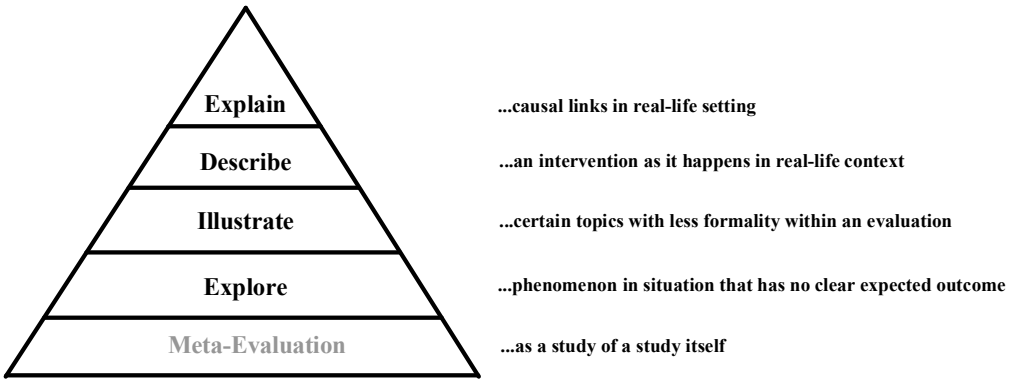

Figure 4. Research applications (Yin, 1994) illustrated as a pyramid.

Yin (1994) defines the case study research method as an empirical inquiry that investigates a contemporary phenomenon within its real-life context, when the boundaries between the phenomenon and context are not clearly evident, and in which multiple sources of evidence are used. Case studies can be single-case or multiple-case studies. The main strength of the approach comes from using

multiple data gathering methods and several data sources. This increases the validity of the research by enabling both data and method triangulation.

A single-case study was a natural choice for the research approach, because a hardware development project in an industrial setting was interested in experimenting with flexible product development. A case study also provided a structure for the research in this situation, because the majority of the data was qualitative and came from multiple sources. The data gathering process is described in more detail in chapter 3.3.

The term “action research” describes a spectrum of approaches to study that focus on research and learning through intervening and observing the process of change. It is a continuous process of learning and change where researchers and clients develop a long-term interest in understanding and resolving a problem or issue (Cunningham, 1997). As a result of this collaboration, the distinction between research and action becomes quite blurred, and the research methods tend to be less systematic, more informal, and fairly specific to the problem, people, and organization for which the research is undertaken. There is no intention, typically, to generalize beyond the particular setting (Patton, 2002). Several authors have described a process for action research. These processes differ slightly, but they all share common characteristics of a cyclic basic routine. Stringer (1999) uses the terms *look*, *think*, and *act* in his cyclic model. It is important to note that action research is a continuous recycling of these activities, not a single-shot linear process. The look stage includes data gathering and building a current picture based on this data. During the think stage, the researcher analyzes what is happening, and explains why things are as they are. The final stage of the cycle, the act stage, represents the action, and includes planning, implementing, and evaluating the action. At the completion of each set of activities, participants will review (look again), reflect (reanalyze), and re-act (modify their actions) (Stringer, 1999). While presented in a simple repeating three-step process, action research in practice can take very complex forms as it is constantly re-shaped according to the real-life setting.

Action research as defined by Stringer (1999) provided a cyclic framework for the study. We were continuously gathering data about the process and the collaboration between the different parties in the project (the look stage). We used this data to analyze what was happening at the moment, what could be the reason for what we were observing, and which areas should be given improvement focus next (the think stage). From the results of this analysis, we identified and planned change actions (the act stage). Similarly to action research, AAgile development also provides a cyclic framework for reflecting on the past and implementing identified actions for improvement. Because of this and the distribution, the teams themselves took significant responsibility for data gathering, analysis and implementing the planned changes. In this process, the author’s role was focusing on being a research facilitator,

observer, data recorder and provider of specific process knowledge at each action research stage. As best practices were not available for applying concepts from Agile software development to hardware development, the experimental and reflective analysis perspective of this process was vital. The author was co-located with the project lead and the project had frequent teleconference meetings, which made continuous refinement possible. Several team members also participated in the deeper analysis of the data with the author throughout the case project, aiming at understanding the project better. This served both goals of the action research; improvement of the performance of the project and contribution to the research.

3.2 Case project

The case study took place in a large global organization with its head office in Europe. In recent years, the organization has invested in setting up several Product Development Sites globally. The organization had an official product development process model, which was an application of the Stage-Gate model (Cooper, 2001). In addition, engineering sites were preparing for CMMI Level 2 auditing. The organization had a dedicated Technology Development Site in Europe. The project under study was initiated to productize a novel technology for creating and managing electricity from the user's actions. The technology was chosen to be used in a wireless building automation system. The technology enables development of battery-less control devices. A business case had been developed, and the complete offer would consist of several devices enabling a simple control system for the European market. Each device required electronics, mechanical, industrial design, firmware and embedded application software development. The requirements were still vague and only a very abstract specification was available.

The project lead worked in Finland. A pragmatic approach would have been to get a co-located team with domain knowledge and experience on the application, but the European engineers with these qualities were tied up in other projects. The teams available for the project were from the company's global development sites in Mexico and China. The developers in Mexico consisted initially of three industrial designers, three electronics engineers and one mechanical engineer. Due to the resource management in Mexico, the developers were only partly allocated to this project, 60-80% of a team member's time. The developers' experience ranged from fresh out of school to five years in the industry. The Chinese developers consisted of four electronics engineers and two PCB layout designers. The average design experience was three years. The electronics engineers were fully allocated to the project, but the layout designers only for roughly 50% of their time. The responsibility for the development of the underlying technology and firmware remained with the Technology Development Site. Technology development could not be seen as completely independent, because mechanical integration was needed. The

initial project organization is shown in Figure 5. The teams were formed based on engineering disciplines. The mechanics team was in Mexico, but the electronics team was distributed between Mexico and China.

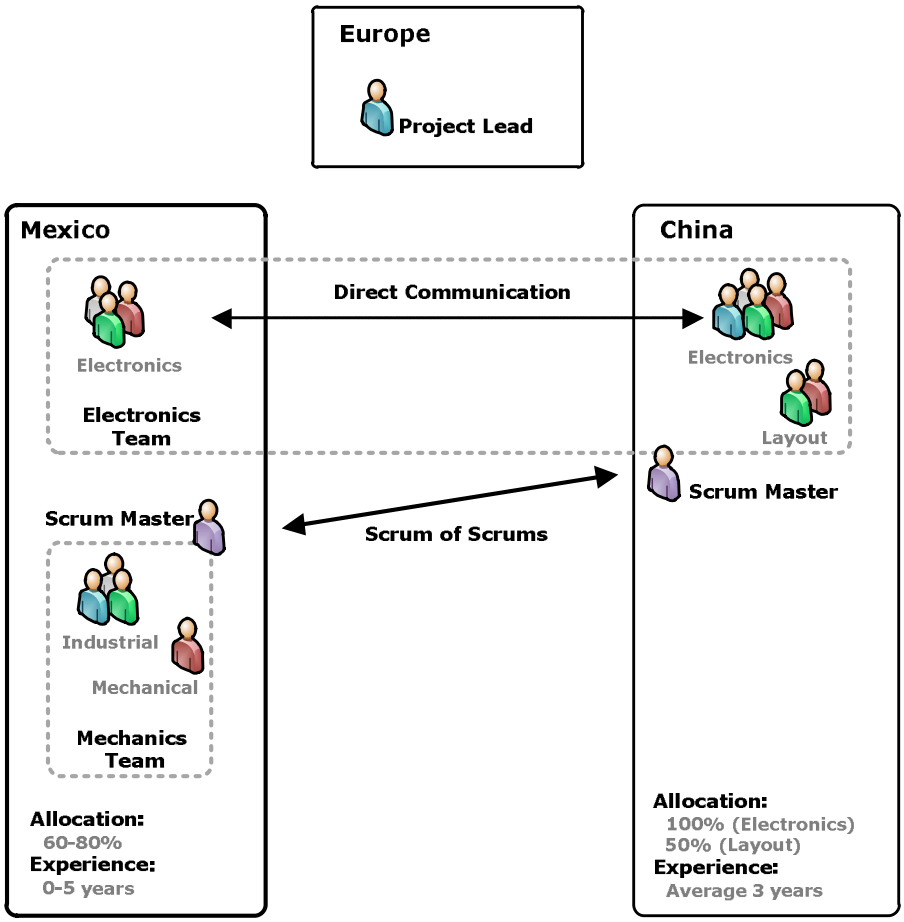

Figure 5. Initial project organization.

The project lead knew that there was a huge amount of learning and knowledge transfer to be done. First of all, the team member profiles he received revealed that many of the development team members were junior developers. Some of them were coming straight from school, ready for their first real project. Secondly, the sites were very far from the targeted European market. It was impossible for them to have the specific market knowledge needed. Moreover, the requirements were very vague to start with. Thirdly, due to the time differences, it would have been highly inefficient for the team lead in Europe to work as a link between China and Mexico. The time difference would have added a delay of 1-2 working days to transferring any piece of information. The project lead was informed by his colleagues that the sites in Mexico and China had been trained in the company's official Stage-Gate process model. They would be accustomed to a bureaucratic, sequential process relying on paper hand-outs – an approach similar to a Waterfall process. Nevertheless, the project lead felt it impossible to execute the project

in this fashion based on the challenges. He had some experience in Scrum with cross-discipline teams developing embedded systems. These projects sometimes had a dispersed team with 1-2 team members being located elsewhere. Distributed projects of this scale were still completely new. There were no guaranties that Agile methods would work, but according to earlier experience with cross-discipline and dispersed development, it was worth giving it a shot. As the project lead said,

"There really were no options according to my knowledge."

At that point, the project lead contacted the author asking for an opinion on applying Agile methods, and for further help training the teams and getting things going. The author had been applying Agile methods to embedded system development for many years, and had previously worked with the project lead.

3.3 Research process and data gathering

Figure 6 gives an overview of how the research process progressed. The left-hand side lists the events of interest from the research point of view. The right-hand side does the same for the project-related events. The project can be seen as having three phases: proof of concept, validation of the architecture and preparation for production. The phases differ in significant ways regarding the maturity of both the product and the project organization. The case project is described in detail in chapter 4.1. The remainder of this chapter focuses on describing the research activities.

A large part of the project organization got together in France roughly half way through the project, as can be seen in the timeline in Figure 6. This event combined the Sprint Review and Retrospective. During the event, 10 semi-structured interviews were conducted. The selection of interviewees was obviously convenience sampling based on everybody's presence. The interviews were short, around 30 minutes each. Interviewees consisted of developers from different engineering disciplines and Scrum Masters. All interviewees were asked to describe problems in the past and possible changes that they have experienced since the introduction to Agile methods. The interview instrument is presented in APPENDIX B. The interviews were audio recorded and full transcripts written up afterwards. After the interviews, we had a workshop with the project lead to identify major themes and categories from the interview data.

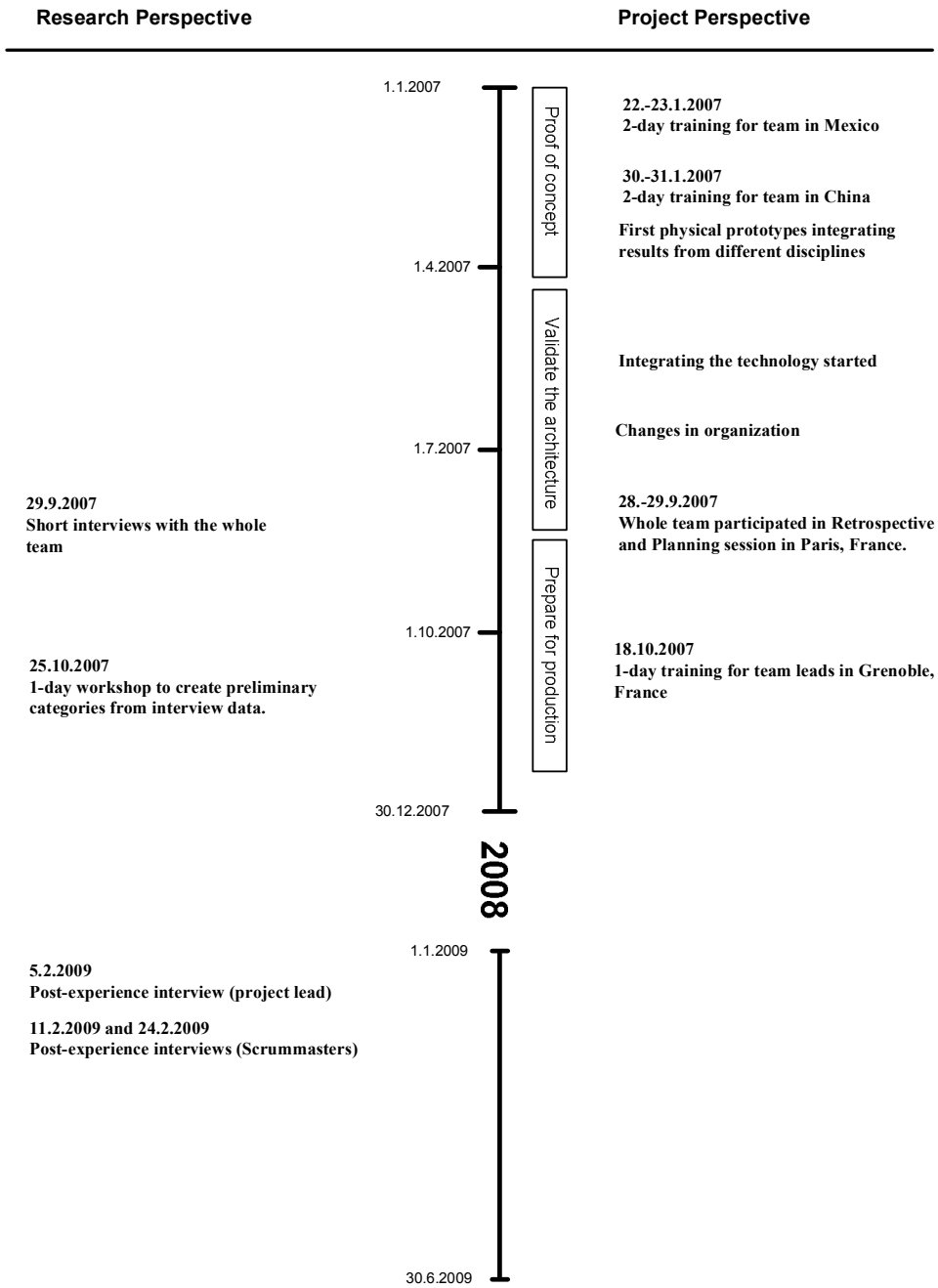

Figure 6. Timeline of events and activities in the project and study.

One year after the project, the author conducted a second round of interviews. The project lead and Scrum Masters were selected to be interviewed based on purposeful sampling. The objective was to pinpoint the issues of interest, and they had the best overall understanding of the project. These interviews were longer, between 70 and 95 minutes each. The interview instrument for the second interview round is included as APPENDIX C. Again, the interviews were audio recorded and transcripts were written up.

In addition to above-mentioned major research events, observations were continuously made during the normal project events. At the end of the project, we gathered and stored all project-related documents regardless of their purpose and format. Eventually, the additional qualitative data consisted of memos, presentations, photos, recorded chat discussions, Product Backlog, Sprint Backlogs, Sprint Review reports and presentations, and team Retrospective summaries. The project lead and Scrum Masters also actively presented their findings to the rest of the organization during the project. These presentations became data for the research. The project's data management system was available for the research as well. Table 6 summarizes the data collection.

Table 6. Summary of data collection.

Collected Data		
Interview round 1	Number of interviews	10
	Interviewee role	Team members
	Length	Average 30 minutes / interview
Interview round 2	Number of interviews	3
	Interviewee role	Project Lead, Scrum Masters
	Length	70-95 minutes / interview
Additional material	Product and Sprint backlogs Records from Sprint ceremonies Other records (memos, photos, etc.)	591MB in total

3.4 Data analysis

Transcripts of all of the interviews were written up from the audio recordings. Analysis of the interviews was done using the transcripts and ATLAS.ti qualitative analysis software. Quotations from the interview data were created and grouped into categories. All categories and quotes under them are fully traceable to individual interviewee statements. Categories had already been created during the project using the data from the first interview round. These categories worked as a starting point. It soon became evident that this categorization was not an optimal fit for the data. The data was now supplemented with data from the later interviews and formal and informal documents and recordings, forming grouped themes. Patton (2002) explains this kind of analysis to be the inductive type. Themes and categories emerge from data, in contrast to a pre-defined framework. Figure 7 is a simplified illustration of the process for the research and the analysis of qualitative data.

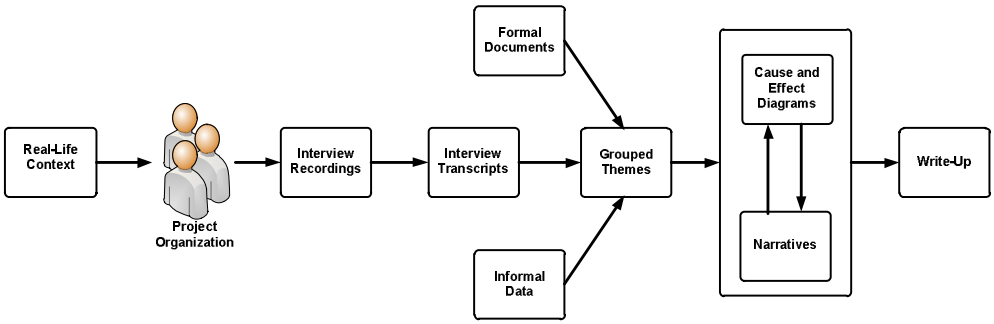

Figure 7. Simplified research method and data analysis overview.

Grouped themes were used as variables when creating a causal network, a cause and effect diagram. Examples of grouped themes are: shared vision and goals, ownership and pride, more committed, continuous improvement, need for new skills, Daily Scrum and time difference. The analysis had a total of 70 variables. Cause and effect diagrams make the cyclical, interdependent and delayed relationships between observations (variables) visible. Narratives were first written using the variables in natural language. This helped in understanding the relationship between the variables, identifying their roles as causes and effects. Having the narrative as an analytical text describing the meaning of the connections is essential (Miles and Huberman, 1994). After the initial version, the narratives and cause and effect diagrams were refined through several iterations. Finally, key findings were identified by looking for enforcing cycles in the diagrams. The key findings are presented in chapter 4.2.

3.5 Threats to validity

The researcher being part of the organization adds a threat of bias entering into the research as early as the data gathering phase. While the author was not an active project member, he was working in the business unit responsible for the project. Coghlan and Brannick (2001) point out two specific issues in the context of researching the author's own organization: 1) clarifying the research project in terms of both your personal and the system's commitment to learning in action, and 2) managing issues of role and secondary access[1]. To help in these concerns, the objective of later writing a case study or experience report was openly discussed throughout the project. On the other hand, this situation enabled in-depth knowledge of the phenomenon to be captured, and this was the priority in justifying the method. Furthermore, the validity of the

[1] Secondary access: access to all specific parts of the organization which are relevant to the research.

data gathering was increased by member checking with several people from the project organization.

The validity of the data analysis had the same threats as the data gathering: misconceptions and misunderstandings, but also bias from the author's own objectivity. To make things worse, mistakes made during the data gathering would accumulate during the analysis. In addition to the earlier-mentioned member checking during the data gathering, the subjects also reviewed the results of the analysis to minimize the risk of any such issues.

4 Results – Case study

This chapter concentrates on presenting the case itself and the results of the study. First, it describes the project in chronological order. The second half of the chapter presents the key findings.

4.1 Case project

An overview of the project was presented in Figure 6, in chapter 3.3. This chapter gives a more detailed explanation of the case project. The description is divided into four phases: the project's front-end, proof of concept, validation of the architecture and preparation for production. The latter three explain the changes in the case project from three perspectives: the emerging product, organization and practices.

4.1.1 The project's front-end

Bootstrapping

It was decided that the author should visit both sites for two days to give a basic introduction to Agile development in general, and the Scrum framework specifically. It was also hoped to conduct the first Sprint Planning. Two days for training and planning was short, but nothing more was possible at short notice, and we needed to get things going. At both sites, the technical project manager was a natural choice to fulfill the role of Scrum Master. Mexico was visited first. They were expected to follow the company's process, but they did not have an established design process at that time. This was contrary to the briefing that the project lead had received before contacting the site. That may have helped with their acceptance of Agile development, because it provides guidance at the team level. After the training in Mexico, the Scrum Master and another engineer also participated in the similar two-day training in China.

While in general the teams liked the ideas of Agile development, there was also some skepticism. When the idea of up-front prototyping was introduced, it was judged to be impossible. The reason was past experience. For example, in Mexico, the prototyping was expected to take three months, because there was a rather rigorous process to be followed when ordering development prototypes. This was much too heavy for an approach investing in rapid and frequent up-front prototyping. In earlier projects, we worked with partners to shorten the prototype cycle. We knew that for simple printed circuit boards we could achieve a prototype cycle of 24 hours from finishing drawings to delivery. We decided that teams would send the drawings to Finland and the project lead would take care of ordering and assembling the prototypes. The prototypes were then distributed around the world using a fast delivery

service. We convinced everyone that we just wanted to test this, and that we need only commitment to trying, not promising to succeed. By doing this, we were able to help the teams to move on despite the hesitation. We decided that the project lead would act as a Product Owner for both teams. Both teams would start having a four-week Sprint length, and Sprints would be synchronized (both teams starting on the same day). We further agreed that both teams would work from the same Product Backlog, and this would be stored as a Excel sheet in the project's data management system.

Working agreement

Guidelines for usage of electronic communication tools for different purposes were set up from the beginning. A wiki was set up for informal communication. The company data management system was also web-based and was opened for the project's QA process related documents. The company's database system for product-related information was used from the beginning for schematics, PCB layout drawings, and product assemblies. The Pro-Intralink system was used to store and share evolving mechanical drawings. Skype was selected to be used for instant messaging. Team leads had a Scrum of Scrums[2] meeting twice a week over the phone.

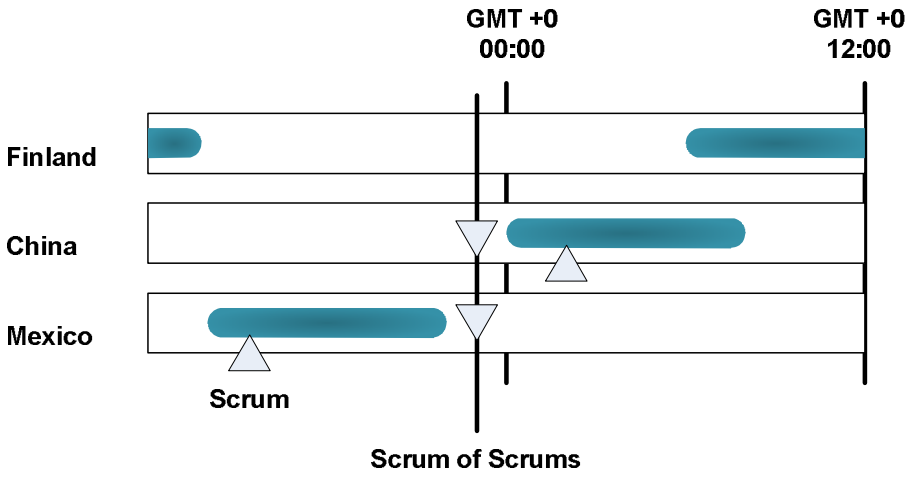

Figure 8. Working hours due to the time difference and the schedule of Daily Scrum meetings.

While all caution was taken to provide good means of communication, we understood that the time difference and other responsibilities sometimes

[2] A Scrum of Scrums meeting is a coordinating practice used when Scrum is scaled to multiple teams. In addition to the Daily Scrums of individual teams, each team sends a representative to Scrum of Scrums meeting to foster knowledge synchronization across the teams (Cohn, 2010).

made it impossible to contact someone for guidance. Without special arrangements, practically speaking, no two teams were working at the same time (see Figure 8). Because of this, we agreed that the site would make any decision locally if issue resolution took more than 24 hours.

4.1.2 First part of project: Proof of concept

Emerging product

In the beginning, teams were encouraged to make the first prototype. At first, they felt uneasy with this approach, because there were too many unknowns. The products were developed for the European market. Many aspects were unknown to the teams, for example regulations and standards and even the size of the products. It was difficult for them to visualize what the project was trying to achieve. The first Sprints consisted of a great deal of information gathering. The team in Mexico, for example, needed some reference designs from Europe to pick up the ideas. Furthermore, for most of the members of the Mexico team, this was their first project with this business unit, first mechanical work including development of electronics in parallel and first project with the site in China. Considering this, it was remarkable that after three Sprints the teams achieved a prototype by integrating work from both teams.

From the first prototype, new prototypes were the goal for each Sprint (see Figure 9 for an example of a set of prototypes after a Sprint). Despite the early achievement, the teams still struggled with the idea of investing in early prototypes. In particular, the team in China did not initially see the value of prototyping in the middle of design, and developers were even slightly frustrated by creating several mock-ups in the beginning. It was found difficult to achieve a significant progress in just four weeks for the demonstration at the Sprint Review. However, there were some very concrete benefits of using the prototype as a measure of progress. The prototype was far from perfect, but it provided an understanding of the magnitude of the knowledge gap. As said above, the teams lacked knowledge on several areas: the market, regulations and standards and even to some degree design for manufacturing. We noticed this challenge early on in the project and tried to offer as much guidance on domain expertise as possible. For example, we agreed to have a local application specialist available on Skype or the phone for the teams.

Figure 9. The emerging product after a Sprint.

Organization

The idea was that the electronics team would be distributed between Mexico and China. This cooperation had problems. The Mexican developers were much less experienced than the Chinese ones, and this resulted in a lack of trust in younger group. Trust was not specifically measured during the project, but blaming between the teams was a clear indication of this at this early stage of the project. Communication between the teams was tried to be enforced by arranging a company cell phone for a Mexican electronics engineer, but the root cause was of course much deeper. Approximately three months after the beginning of the project, the electronics design responsibility moved completely to China, and the electronics engineers left the project in Mexico.

Practices

The inspect and adapt cycles are the focus of Scrum. The idea is that the team continuously reflects on its performance and seeks improvement. The basic practice is the Retrospective meeting where the team focuses on improvement initiatives to be implemented in the next Sprint. It was difficult to observe the Retrospective meetings because of the global distribution, but what the author and project lead observed during the first couple of Sprints was that we did not hear anything about improvement. We did however hear some complaints about practices that were not working (examples are given below). We emphasized the importance of holding Retrospectives, and proposed that a short summary of Retrospectives should be given during the Sprint Review meeting.

The Scrum Masters in both teams were disciplined to facilitate all the Sprint meetings. The Sprint length was 4 weeks, and the Sprint rhythm was synchronized between the teams. The Sprints ended on time (teams honestly

presented what was done by the Sprint end) and both teams were committed to meeting the Sprint goals.

Product Backlog was a simple Excel sheet. Each team had their own Sprint Backlog, which was also stored in Excel. All Backlogs were distributed in the project's data management system. The Product Backlog was organized into releases, each with a release goal. The first release goal for a release combining three Sprints was very abstract:

"Proof of Concept with [technology] and ceiling relay."

There was no Sprint goal defined, but Sprint Backlog Items (SBIs) were defined on fairly high level and were quite unambitious, for example:

"Schematics Plan for [technology]. "

From the beginning, both teams used relative user story points (Cohn, 2006) for estimating the size of the work, and defined acceptance criteria for each Backlog item. For example, for the above SBI, the acceptance criteria was:

"Draft (block diagram) to identify the job to be done."

In the next Sprint, the SBI and the acceptance criteria were:

"Schematics for [technology]."
"Schematic enabling PCB work."

At this point, the development process was more like a mini-Waterfall with sequential tasks. The aim was toward more parallel work, but we did not want to introduce too much demand for learning at the beginning. It was also difficult to guide the planning, as the teams and their capabilities were unknown. On the other hand, the teams were new to this kind of planning and did not have experience of the vertical slicing[3] of work.

The project lead was the Product Owner for both teams. He participated in Sprint Reviews of both teams, but they were held separately for each team. Sprint Reviews were conducted using teleconferencing and NetMeeting. Teams presented their achievements using PowerPoint presentations of the drawings. At first, the presentation was given by the Scrum Masters, but then the developers were given their voice. This was considered to be a good thing among developers. It gave an opportunity to take pride in their own

[3] Vertical slicing means splitting a larger work item in such a way that completing each split work item requires observable progress in all components of the product (Ratner and Harvey, 2011).

achievements. After the third Sprint, the prototype was ready in Finland and participants could follow the presentation, but also have the concrete prototype at hand. Participants included stakeholders from the technical and business perspectives. However, this was not always easy. The project was not following the standard process of the organization and was thus "unofficial." This made it challenging to engage stakeholders from the other functions. The project had a Finnish product manager that was interested in both the emerging products, and the new approach to developing it collaboratively with business. He was present at the Sprint Plannings and Sprint Reviews. The concrete feedback he could provide was appreciated. He also understood the principles of Agile prioritization and scope management to meet the deadlines. He was able to make the necessary trade-off decisions to focus on the minimal marketable feature set. The technical project manager from the Technology Development Site also attended Sprint Reviews over NetMeeting.

One significant concern reported by the Chinese team's Scrum Master was that the team was struggling with the Daily Scrum. The developers thought that it was enough to give the schedule, and it was not necessary to follow up daily. The result was that the Daily Scrum was not attended, and the Scrum Master needed to gather people, or even walk to each person's desk to gather the information. At this point, it was not understood that the Daily Scrum is an information synchronization event between team members, not an activity reporting event for the Scrum Master.

4.1.3 Middle part of project: Validation of the architecture

Emerging product

The middle part consisted of two releases, each having three Sprints. While the first part focused on proof of concept with two units, providing teams with knowledge on the domain and application, the middle part aimed at widening the offer. The second release, for example, had the release goal:

"Enable Complete Market Launch."

This goal was understood as to limit the offer to a minimum marketable set of devices, and this set was to be prototyped close to industrialization. The middle part introduced several new devices, which of course used ideas from devices done earlier, but the overall architecture also started to emerge. When a new device required a different architectural concept, in many cases it was necessary to redesign the existing prototypes to some degree.

The prototype level shifted from learning prototypes (which were later called "first-level prototypes") toward functioning prototypes more ready to be installed and presented to a wider audience. This change made new kinds of issues visible. The teams did not have 3D models of electronic components.

The PCB layout was only done in 2D and design conflicts became visible in the prototypes. There was a lack of understanding of production engineering in general. For instance, it was not clear how the PCBs need to be supported. The positive side was that this feedback was very concrete and real. The first part of the project focused on basic skills in up-front prototyping. Now it was time to learn how to use the information from prototypes and feedback.

Organization

Half way through the project, a more experienced electronics designer joined the Mexican team and a mechanical designer joined the Chinese team. They were both understood as *communication bridges* (see chapter 4.2.2), and were identified as the key reason for accelerated development. This was experienced as a real help in integrating the work of the teams. Trust had emerged between the teams. The new project organization is illustrated in Figure 10. In contrast, communication was affected negatively by the long vacation period in Europe. This was problematic for the development teams, especially the Mexican team. They were already used to moving really fast, but when they lacked the feedback, they experienced the slowing down as demotivating. They were empowered to make decisions, but if they were unsure they needed to wait for certain decisions to avoid the risk of going too far in the wrong direction.

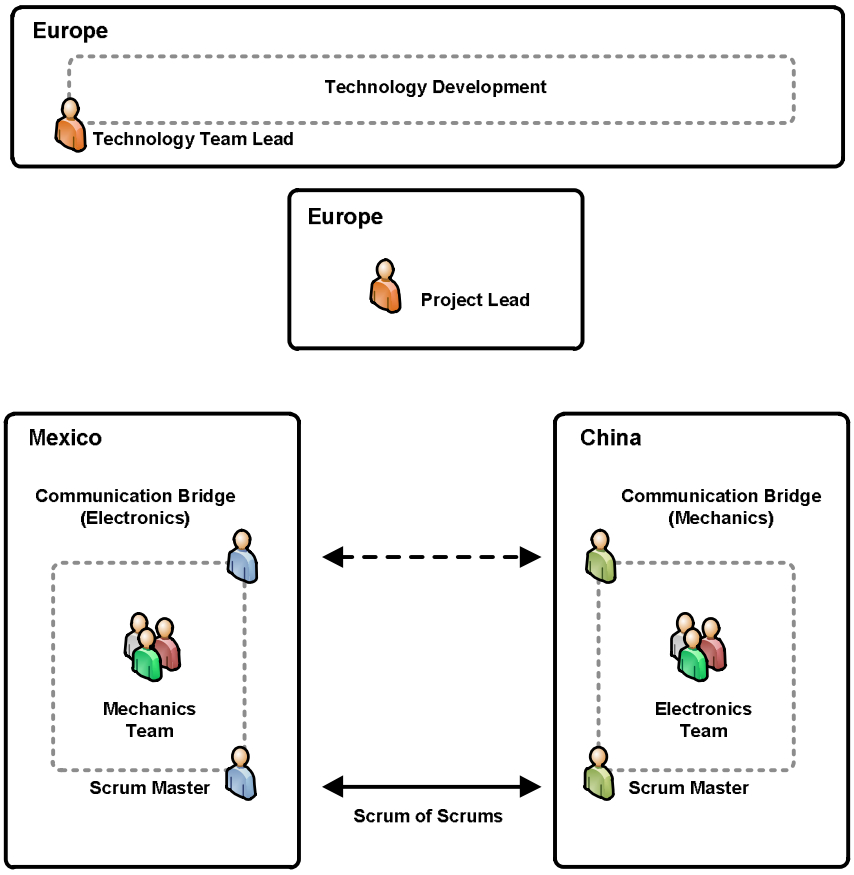

Figure 10. Global project organization and communication mechanisms.

Firmware development was done at the Technology Development Site in Europe along with maturing the technology. The teams used the Product Backlog to synchronize their requests regarding the technology. These items in the Backlog were estimated as zero effort from the product development teams. From time to time, members of the Chinese team needed to make changes and debugging to the firmware development. This was particularly needed because the European team was not working with the incremental rhythm of the teams in Mexico and China. The Technology Development Site was considered more like an external supplier rather than the third team. Furthermore, they behaved negatively toward the rest of the project organization. This often came out as blame, starting the feedback with, "They are not competent." The negative feedback was targeted at different organizational units, and even persons. In addition, they kept promising the technology would be ready, but reported continuous delays. The lack of trust identified between the teams in the early phase was now visible in the relationship with the Technology Development Team. The technology team lead discussed directly with the Scrum Masters, but it was not experienced as a trusted equal relationship. Rather, the Technology Development Site was experienced as trying to command and control the project. In retrospect, the project lead felt that he should have monitored and guided the co-work toward collaborative behavior. The problem with mixing Agile and Waterfall delivery is illustrated in Figure 11.

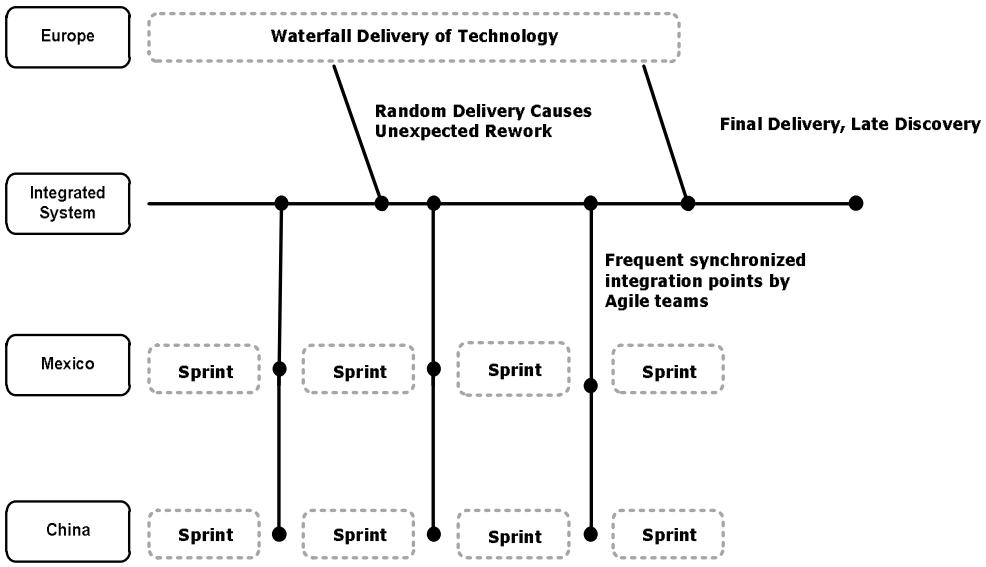

Figure 11. Conflict between Agile and Waterfall delivery.

Practices

At this point both teams were comfortable with basic Agile and Scrum practices and small adjustments were made continuously.

The core Scrum meetings remained as in the beginning. The Sprint length continued to be four weeks, and the teams were more comfortable with this.

Backlog practices had started to evolve. From Sprint 6, the Product Backlog was modified to have a separate column for each team's estimates. This was needed because while both teams used story points for estimating, they did not have a common reference. Naturally, their velocities also varied widely. Product Backlog items were not interchangeable between the teams, as they were based on different engineering disciplines. The single, shared, Product Backlog was nevertheless used to create a rolling-wave plan, keeping the teams synchronized and focused on the same higher-level goal. After the Sprint Planning Meeting the teams did an engineering task breakdown at each site. Both teams started to use a physical Scrum Board to do project planning and monitoring during the Sprint more informally. For mechanical work in particular, there were many dependencies between different tasks. We of course tried to enforce figuring out as independent tasks as possible, but on many occasions this was impossible. The Mexico team developed a method for marking these dependencies, using colored Post-Its to indicate tasks that belonged together. If there was a certain order of those tasks, they numbered the tasks indicating the order. Both teams also used swim-lanes as a method of grouping items, see Figure 12.

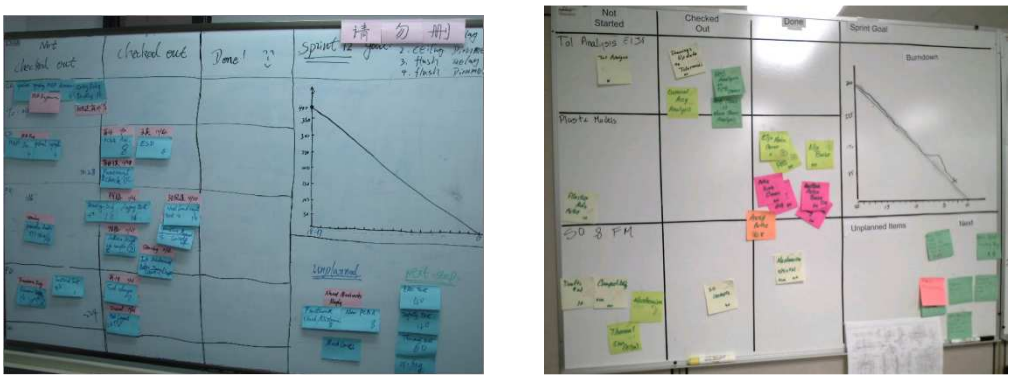

Figure 12. Physical Scrum Boards of teams in China (left) and Mexico (right).

During this period, the Product Owner realized how much work the Product Backlog needed. He was also the Project Lead. This took time away from managing the Product Backlog and it started to show in the quality of the Backlog. For this reason, we agreed to have a Product Backlog workshop with all three parties, Europe, Mexico, and China. This helped the project, but Product Backlog management remained an issue until the end of the project.

4.1.4 Last part of project: Preparation for production

Emerging product

The project started to focus on production quality, and again new challenges were revealed. Many of the designs still had remaining weaknesses. Issues identified in the Sprint Review did not get handled during the next Sprint. The project lead thought that this may be caused by the attitude toward prototyping. Jeopardizing the quality of prototypes was considered acceptable, even normal: they were "just prototypes." On the other hand, prototyping needed effort from the project team, so they were not eager to prototype a minor change. In addition, because mass production quality was the goal, the batch size of prototypes was growing. This caused even more overhead. As an end result, a large amount of work remained that was not visible to everyone.

Only toward the end of the project could the Technology Development Site start to provide fully functioning samples using the novel technology. This also resulted in some additional design work. While the prototypes had been there throughout the project, they were known to be partly designed using borrowed technology. The technology that was said to be ready a year ago, and which was promised to be ready soon throughout the project, was finally there. This was one of the reasons for the difficulties in finding the stakeholders earlier in the project. The shared trust was missing.

Organization

When it was time to start thinking about industrialization, it became evident that the Technology Development Site lacked experience of the design process for manufacturing. A Mexican designer traveled to work in France for two weeks, mainly for tolerance analysis of mechanical parts. Due to the above-mentioned lack of trust, he was directed to conduct the analysis in a way that contradicted his own expertise. As a result, he conducted two different analyses. When the first mechanical parts arrived, his original analysis was proven to be the correct one. This lack of trust remained between Europe and other sites throughout the project, and the project suffered from this.

The teams in Mexico and China were now used to very open communication between each other, and it was hard for them to understand the reasons for talking to team members through the team lead in Europe. Concrete evidence of lack of understanding and weak communication was provided by the fact that the technology team only realized at the end that the project actually worked on several products, not just one. In this light, it is comprehensible that they did not always understand the questions from Mexico or China.

On the other hand, observing the Mexican and Chinese teams now was very different from the beginning. The teams were not competing with each other, but working together toward the same goal. While it was understood that

collocated working would be easier and more efficient, the counterpart was not blamed for this. There was mutual trust. The trust had emerged throughout the project, but there were some key enablers, such as everyone being perceived as equal contributors to the shared goal and conscious avoidance of blaming.

Practices

It was decided to have a larger Retrospective meeting and get everyone in the project organization together to reflect on the past, and create a shared understanding of the project's direction. A meeting in Europe was scheduled at the company's premises, which was convenient for everyone. The Retrospective meeting was held on the last Sprint, but also on the whole project so far, to map the bigger picture. Several problems were identified during that day together. As an example, one team had been using outdated versions of drawings from the database. A simple naming convention was agreed on to avoid this. The greatest achievement of this Retrospective was the decision to look for local partners for prototyping. This did not work in Mexico because the supplier was very expensive, and the quality poor. The saved time compared to ordering mechanical parts from Finland was spent in sanding the parts. However, in China, it was possible to find a reliable supplier for both electronic and mechanical parts. This cut down the cycle time from drawings to testing. A negative side effect was that managing the prototyping took more time from the team. The conflict between the project management approaches of the Technology Development Site and the rest of the project organization was also discussed. As a result, Agile development training for the Technology Development Site was agreed. The one-day training was well taken up, but the case project was too far advanced and this did not affect working habits.

At this time, the teams also identified a need to do something about the Sprint length, which had been four weeks. The planning had changed to include several different time spans, as the individual devices kept maturing, but the system architecture affecting all devices was also emerging. Several devices were maturing at the same time, but at different levels, requiring different amounts of precision. Because of all this complexity, teams felt that four weeks was too long to get feedback. They did not move into actually shorter Sprints, but they scheduled an "Intermediate Review" in the middle of the Sprint. This event was even more informal and only involved the project lead. The normal Sprint Review was still held at the end of the Sprint. Changes outside the project organization also seemed to have a negative impact at the development team level. According to one Scrum Master, there was a clear change in atmosphere in the last Sprints. Earlier planning had been more open and based on what the team felt was possible. Coming to the end, they experienced more pushing to fit a larger scope into the Sprints.

Backlog practices kept evolving. Items in the Product Backlog were defined on a larger scale for the last Sprints. This was possible because the teams had proven their capabilities to estimate and manage their work, even based on fairly abstract goals. They were always able to get the details for the feature at the moment when they started to work on it. Estimation was still done using relative story points, but it was estimated together with members from both teams. For example, Sprint 10 had a goal:

> *"Ceiling Mounted at production level."*

An example of a Product Backlog Item identified for this goal was:

> *"Ceiling-Mounted full functionality and installation with [technology] prototypes."*

The Product Owner's role was challenging at this point. The official objective of the project had changed to proof of concept instead of actual commercial launch. This had also lowered the priority and it was difficult to involve people outside the development organization. He still acted as a representative of the customer and business toward the team, but it was difficult to get real feedback on his work from outside.

When China took responsibility for their prototyping, they also started a practice of taking a video of the prototypes and distributing it prior to the Sprint Review meeting. This worked very well, as the quality of the pre-sent video was obviously much better than the live video using NetMeeting, and it gave attendees a chance to familiarize themselves and prepare questions and feedback prior to the meeting.

The mechanical team in Mexico also experimented with pair design. In retrospect, this was considered a good technique for creating ideas and sketches. Building the actual assemblies was considered to be more effective when working solo again.

4.2 Key findings

The findings are grouped into four categories: accelerated learning, improved communication, improved commitment, and remaining and new challenges. They are presented in the following sub-chapters. Each sub-chapter begins with a table presenting the summary of the category. The summary includes the name of each finding, a description of the finding and the author's recommendation for the future based on the finding, but also on information from other sources outside the case study.

4.2.1 Accelerated learning through up-front prototyping

Table 7. Summary of accelerated learning.

Finding	Description	Recommendation
Up-front prototyping	Early prototyping was used to accelerate learning and to provide proof of the chosen concept. Early physical prototype provided several additional benefits: trusted measure of progress, creating the shared goal, and creating a stronger buy-in among the stakeholders.	Prototyping introduces overhead. For this reason, communicate the expected benefits clearly, streamline your prototyping process and maximize the use of prototypes.
Failure is an option	People were encouraged to take the initiative and make decisions locally. If they failed, this was considered a good thing working as a catalyst for learning.	Great care should be taken to develop an environment where it feels safe to fail. For knowledge creation, it is important that the outcome of the prototype cannot be fully anticipated.
Feedback	Feedback from the product and technical management was available for the teams. It was seen as valuable, and it contributed significantly to fast learning.	Provide rich feedback for the development team, both from a business and a technical perspective, and make the feedback cycles as short as possible to maximize learning. Incremental development does not have a detailed plan to provide control, thus control is needed in the form of feedback.
Emerging product	Different products and the system matured incrementally and iteratively, while the project's focus shifted. Terminology developed to describe varying prototype maturity levels. At times, teams missed acting on feedback, and the unresolved issues resulted in uncertainty about the work needed when moving into production.	The maturity of prototypes should improve continuously throughout the project. Experiments should be narrowly focused to acquire new knowledge, and this knowledge needs to be used in consequent prototypes. Experimentation should be steered by a clear vision. The path toward the vision, however, is continuously redefined based on learning.

Up-front prototyping

Description

In this project, prototyping was used for learning, not to validate existing knowledge. What is traditionally negatively called rework, or scrap work, was now considered valuable. This is called up-front prototyping, and is enabled by today's design tools and fast prototyping technologies. In the case project, up-front planning seemed completely irrational, as the knowledge to base the plan on was non-existent.

Having only vague requirements is not uncommon at all according to Thomke and Reinertson (1998):

> *"One of the authors has worked with hundreds of product developers and has yet to find a single project in which the requirements remained stable throughout the design. Surveying more than 200 product developers over the past five years, he found that fewer than 5% had a complete specification before beginning product design. On average, only 58% of requirements were specified before design activities began. The inevitable result is changes."*

Experimenting with physical prototypes truly accelerated learning in the case project. The first prototypes were assembled after just three Sprints. They revealed the magnitude of the gap in domain and technology knowledge. This challenge became evident in a matter of weeks. In the past, issues like this had remained unrealized until the end of projects, for half a year or even more. Domain knowledge was quickly transferred by providing direct access to domain experts exactly when the information was needed (see Pull information in chapter 4.2.2). The project applied "Just-in-Time" learning (Hutchings et al., 1993) for technology and domain knowledge by letting the developers try out the design with current knowledge. Learning was very fast when the receiver of the new knowledge had the need for it in their daily context. Early prototyping also had several other positive effects. The physical prototypes provided a very reliable measure of progress. Because prototypes needed input from both teams, it can be argued that this contributed to trust building by creating shared goals and objectives. In addition, stakeholders got to see the evolving products, and it created a stronger buy-in.

> *"I feel that it's much better to have shorter cycles to actually have something physical for people to see. That way it's available for all, not only marketing. It's better than not achieving anything."*
>
> *Developer, Mexico*

The downside of frequent prototyping was the cycle cost. Working with physical prototypes introduces costs in multiple ways. Design documents need to be prepared for the prototype supplier. The prototype supplier has material and labor costs when the prototypes are assembled. When the prototypes arrive, they need to be tested and some rework may be required. The cycle cost is illustrated in Figure 13. Having far more prototyping rounds compared to the traditional process model caused several of these cycles in a short period of time. This was mentioned in several interviews.

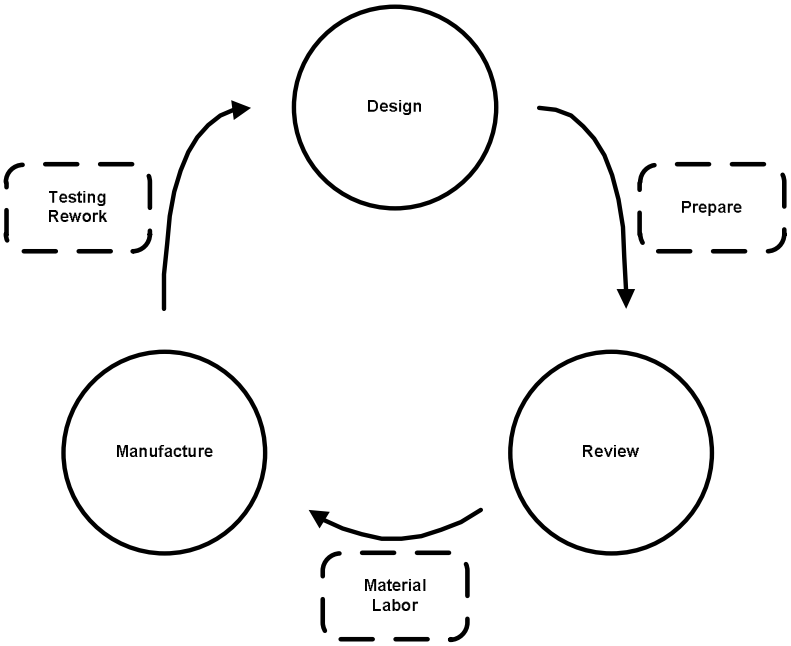

Figure 13. Cycle cost of prototyping.

Recommendation

To address the concerns regarding cycle cost, the many benefits of early prototyping need to be communicated to developers. On the other hand, the overhead can be reduced by developing the prototyping process to be more suitable for frequent prototyping (see chapter 4.2.4: Large-scale organizational change is needed and Need for a change in engineering practices). Despite our effort, cycle cost remained. Therefore, it is important to maximize the value of prototypes. Prototypes can be used to replace traditional progress and status reports, and to gain stronger buy-in across the organization.

Failure is an option

Description

As early as the first interviews, several interviewees mentioned that in this project, it felt like it was alright if the developer did not get it right the first time. Junior team members were encouraged to experiment through communicating that failure is not to be blamed. This is very contrary to the so-called get-it-right-first-time approach which tries to minimize mistakes. The first prototype in this project was known to have many shortfalls. At first, the engineers did not see the value in building a prototype which will likely not work. However, it was extremely valuable in providing information about areas where knowledge was lacking. A developer explained:

> *"I like the Agile process. I think it's the way that we have to design, because experimentation is not making mistakes. I feel that in the other projects, they are scared of getting something wrong, so they inspected all the possibilities before saying, 'Yes, it's OK. You can make a prototype.' In [design brand], I think we made a lot of mistakes, but they are happy mistakes because you learn from them. So I like the freedom that we have. The process, the Sprints, makes the design process faster."*
>
> *Developer, Mexico*

Recommendation

Failure is the single most effective way of learning. Create an environment where developers feel safe to fail. To avoid calling experimentation a failure which has a negative sound to it, one could think of adapting a new 5-point test for failure. The test is adapted from Hamel and Prahalad (1994):

- Did we manage the risk appropriately?
- Did we possess reasonable expectations about the solution?
- Did we learn anything?
- Can we quickly react, and try again?
- Do we still believe that the opportunity is for real?

Failure should be declared only if the test has just NO answers. Donald Reinertson presents a testing strategy for maximizing new information creation in his book *The Principles of Product Development Flow* (2009). The strategy is based on the idea that an event contains more information when it is less likely to happen. In development, this means that if we are pretty sure that our design works and after testing we are convinced it works, our information creation was next to nothing. Reinertson suggests that we define a test strategy aiming at tests that have a 50% probability of success. This maximizes the creation of new information. In this strategy, failure is as expected, but also as welcome, a result as success. "Our testing processes need to have an adequate failure rate to generate sufficient information" (Reinertson, 2009).

Feedback

Description

We enforced the continuous involvement of the business side in the project. Because of this, product management representatives attended the review meetings and planning sessions until the end of the project. Technical managers participated as well. The feedback they were able to give, and the reasoning for prioritization, was highly appreciated by the teams.

> *"...also the monthly review, when you finish the Sprint. They also give us a lot of feedback so it's great, because we can improve more, get things better and faster. If we wait until the last day (duration of 2 months) and see a problem, you cannot fix it the last day."*
>
> *Developer, Mexico*

In addition, the project lead was able to provide feedback from the customer and technical points of view on day-to-day questions. The feedback the team could get from many different angles was one of the key contributors to fast learning, especially at the beginning of the project. The need for feedback in this type of development was evident. For example, during the European summer vacation, the teams experienced the lack of feedback as demotivating.

> *"..., we were moving very very fast and then we hit European vacation and there was no information, no feedback, there was nothing. We were waiting two months for information.... Especially with industrial design was very hard to come up with a solution because it had a lot to do with European point of view. We were aware of that."*
>
> *Scrum Master, Mexico*

Toward the end, the changes on the corporate level adjusted priorities and the project was lacking a "real" business and customer role. The project lead in particular experienced this as demotivating. In the post-project interview, he questioned whether at that point the project should have been put on hold.

Recommendation

It is important to provide as rich feedback for the project team as possible. To accelerate learning, the feedback should be available from both a business and a technical perspective. Furthermore, the shorter the cycle from experimenting to receiving feedback is, the faster the team can learn.

When the experimenting drives the design, instead of the up-front plan, it is important to replace the plan with continuous feedback. If you do not have a plan and you cannot get feedback, you will be lacking all of the controlling elements in your process.

Emerging product

Description

On the higher level, a certain sequential process characteristic was identifiable. Different sequences are described in Table 8. These sequences were also presented in the case description in chapter 4.1. However, each sequence was executed through several iterations. While the overall goal or the primary focus changed clearly between the sequences, the detailed planning for the ways and scope of development was adaptive. In practice, the change in the focus between the sequences was more of a smooth shift than a single event.

Table 8. Sequential phases in the case project.

Phase	Timing	Focus	Activity in the case project
Front-end	Before Project	Training / setting the mission	Hands-on training on new method was provided for both teams. Teams were involved from the beginning in defining the process.
First part	Sprints 1-3	Selection and proof on concept	Up-front prototyping was used to achieve early victory and clear visibility of weak points, for example, lack of domain knowledge. Concepts were proven with non-perfect "first-level prototypes."
Middle part	Sprints 4-6, and Sprints 7-9	Validation of the architecture	Focus shifted to "functioning prototypes". A system architecture evolved between different devices. Often prototype goals were over-ambitious. For example, an improvement in architectural concept was tested on the next device. This meant that two goals were trying to be achieved with one prototype cycle.
Last part	Sprints 10-12	Prepare for production	Industrialization for the mass market requires a large amount of compromises from several different disciplines. The team started to feel the schedule pressure. Minor issues left undone became visible at this point.

The project had its first physical prototype after just three Sprints. After that, the prototypes followed to mature the design iteratively and incrementally. Teams developed their own terminology to describe the level of prototyping: first-level, and functioning prototype. The first-level prototype was typically not intended to work perfectly. Sometimes it was not expected to function at all. It was just for testing an architectural concept, for example. In contrast, a functioning prototype was expected to function properly while unsolved issues could remain in some areas, such as integration with plastics and electronics, or maturity of plastic and mechanics parts. In the post-project interviews, all three interviewees were able to describe what these different types meant. Having this terminology is a great help to ensure that everyone understands the expected maturity level and what the current prototype is trying to achieve.

There was still something missing. During the Retrospective meeting in Paris, a developer from one team approached the author in private:

> *"I have a question. How do we move from just prototypes to production level?"*
>
> *Developer, China*

It was not clear what was missing from the functioning prototype in terms of production quality. The terminology had evolved during the project, so it did not cover the product until production. Defining the levels of prototyping showed promise, but it should be done for the whole life cycle. One contributor to this gap in maturity was the fact that this project reached a point where it felt like "prototyping for the sake of prototyping." When you get the sense of speed during fast prototyping, it is easy to get carried away. In the case project, there was sometimes too little time for taking lessons from previous increments into account. Identified issues were not taken care of. On the other hand, occasionally there prototypes were intended to solve too many uncertainties at once. This made testing more complicated.

Recommendation

The prototype as a term has a long history, and is often used in context of "it doesn't matter, it's just a prototype." When you use the prototype as a measure of progress, the maturity of the product should improve almost linearly throughout the project. An early prototype could be done with a larger PCB or expensive integrated circuits. The first prototype might be far away from the market for a simple reason like cost, but nevertheless, it gives a rough idea of many parameters, such as bill of material and power consumption. The testing strategy is different for each phase and each part. Some parts can be thoroughly tested for validation early on, while some are just tested for creating information. The main point is that the design matures continuously. There should not be a single huge leap from prototyping to production.

It is recommended that prototyping is based on a clear goal. This goal is preferably kept quite narrow. Clay and Smith (2000) recommend using the prototypes to answer specific questions independently. When we obtain answers, it is important to transfer the knowledge from the earlier prototype to the next spin. Remembering the work left undone, or finding a fix for a problem, becomes more difficult the longer the time between finding the issue and dealing with it. The Agile literature uses "Technical Debt" as a term describing quality issues that are not taken care of; for example, long functions, architectural violations, or duplicated source code. Having too much technical debt can lead to a situation when you find new issues faster than you can fix them. Leaving small issues unfixed until the final prototype before production resembles technical debt.

A strong shared vision for the outcome is needed to keep the development inside the preset boundaries, even while you adapt your plan according to the new knowledge. Without these boundaries, there is a risk of endless series of experimentations moving too far from the original goal. In other words, keep thinking about the big picture, but narrow the scope for a Sprint to focus on knowledge creation. For example, you need to keep thinking about a whole product family sharing the architecture, but in an increment you can focus on a single product according to the existing knowledge. You can then use the acquired learning to redefine the big picture. It is true that in less flexible engineering disciplines the dependencies are stronger than in software development, but it is possible to plan the project simultaneously at different levels of abstraction.

4.2.2 Improved communication

Table 9. Summary of improved communication.

Finding	Description	Recommendation
Trust	Trust played a major role and affected many other issues in the project. In the beginning, trust was not established inside the distributed electronics team. Later, trust was built with an experienced Mexican engineer who was able to contribute to the shared goal.	Trust needs conscious attention at all phases of the project. An environment in which blaming is avoided is essential.
Pull information	Team members were active information searchers. They were supported by a network of specialists to provide the information exactly when they needed it. Scrum Masters held Scrum of Scrums meetings twice a week to pull information from the other team.	Communicate early the idea of pull information to teams. This may need coaching, as asking for help may be considered as a weakness. Create networks of expertise in advance.
Communication Bridge	Teams were formed by engineering discipline. Teams developed a practice they called Communication Bridge. The Communication Bridge is an engineer who represents the engineering discipline of the other team. They help the team to interpret the other team's perspective.	It is always better to establish cross-discipline teams. If this is impossible, make sure that each team has access to a representative of each engineering discipline.
Technology	Many different tools were tried out. In the end, email was considered to be the most useful after some ground rules for effective use were established.	Create a working agreement on the usage of tools, such as naming conventions, email behavior, etc. Encourage experimentation with tools.

"At the end it comes down to a way you want to work. If you really want to work with the team, it doesn't matter if you are half a world away."

Scrum Master, Mexico

"The frequency and quality of communication is much higher for [Case project]. I think it's due to the contribution of all members in all teams. Everyone works hard and pays attention to the problems of others and is quick at giving feedback."

Developer, Mexico

Trust

Description

The best way to build trust at the beginning of a project is to meet face-to-face (Paasivaara, 2005). In the case project, not all of the team members from both teams met one another. In the beginning, two members of the Mexico team attended the first training and planning session in Shanghai, China. They stayed there for a couple of extra days to get to know the Chinese developers, and to create the initial plan for engineering tasks for the first Sprint. During the project, the meetings were scarce, and even the Retrospective Meeting in Europe only involved part of the Chinese team. However, both teams had a Scrum Master that was trusted, and the Scrum Masters had a good relationship.

The distributed electronics team struggled with building trust initially. Developers at both sites were new to this kind of working and there was a level of competition between the development sites. The team members in China were more experienced. These reasons, along with all the other challenges teams were facing at that time, may have ignited the problems with co-working. Later, when the Mexican team was joined by a more experienced electronics engineer, trust was built without meeting face-to-face. He was well respected, because he could provide value to the design. The teams came to call this practice a Communication Bridge (see below). The other party was seen as an equal contributor to the common goal, the project. This was enough to build a strong bond between teams.

"Personally, what didn't work for me is that I never felt like I was part of the team …I didn't feel that I was treated as an equal. Probably the team didn't have the need for an electronic engineer. I know it worked out quite well for [Communication Bridge, later]. I think they really needed him."

Developer, Mexico

There were of course many other elements fostering trust and team building, for instance, shared short-term goals between sites, conscious avoidance of blaming, and synchronized product demonstrations between the China and Mexico teams. Teams were empowered and encouraged to make decisions by themselves if the project lead was not available in 24 hours. These decisions were not criticized by the project lead, even if they turned out to be wrong, and this built trust between the project lead and teams. The Scrum Master in Mexico explained this in the first interview during the project:

> *"…you know when we find a mistake from my part, he [project lead] says 'no, no, it was my fault, I was not paying attention to what you showed me'. Or something like that. I think it is good. Because it is not about whose fault it is, but how to solve it."*
>
> *Scrum Master, Mexico*

The Scrum Master in China followed similar lines:

> *"… [Project lead] always said OK if you don't do that or that, it is your team and you do your team plan. I trust your team. This was always the same voice we heard from [Project lead]. Also for the demo, or Sprint Review, [Project lead] said the same, like if the team says we had this kind of difficulties, we didn't achieve what we planned in the beginning of the Sprint. [Project lead's] comments were always like 'OK, it's like this, you did a good job, …you did what you demoed and we need to do a Retrospective on how to improve, but great work. Thanks to the team.'"*
>
> *Scrum Master, China*

Recommendation

Trust plays a significant role in any project's success. Agile methods have many trust-building elements built in, such as shared goals and responsibilities, but it does not happen automatically. Furthermore, it is not just that building the trust is hard - losing already-achieved trust is surprisingly easy. It can happen with a missed reply to a single email. For these reasons, trust needs attention throughout the project. Avoiding blame, and instead focusing on solutions together, is one of the most powerful techniques for this. "Leader's role in this is to safeguard teams from a blaming mentality so that they can transform into an action mentality. To accomplish this, the leader should shepherd teams from a practice of discussing blame to a practice of discussing solutions" (Tabaka, 2008).

Pull information

Description

The overarching idea for communication management was to create teams that actively sought the information they need at a given time. The objective was to make this information easily available. In the beginning of the project, domain and regional standards knowledge were taken for granted by the European leadership. The need for knowledge transfer became visible through trial and error (see Up-front prototyping, chapter 4.2.1). The lack of domain and application knowledge could have triggered different behavior, for example, blame for being incompetent or massive pre-specification. Instead, the project teams were allowed to experiment and ask for more information when needed (see Failure is an option in chapter 4.2.1 and Trust in chapter 4.2.2). The project lead did not decide when and what information to send to the teams, but rather made all the information easily accessible by request. Jim Highsmith differentiates the two approaches as push and pull information respectively (Highsmith, 1999). As an example of pull information, anytime an engineer needed help on detailed technical issue, he was able to contact experts in Finland directly. The pull information approach was also enforced through Scrum Masters having a Scrum of Scrums meeting twice a week by phone.

> *"When I saw an opportunity for self-organization, to activate the development teams to pull data, it felt like it could work. We shouldn't even try to control it, but just give them a target and activate them in gathering information."*
>
> *Project Lead*

Written specifications were scarce during the project, but when there was a clear need for a document, it was created. A good example is the creation of a specification for so-called "push and push" – the concept of pairing two devices in the wireless network. The general principle was given in a matter of minutes while the implementation was being planned, in the Sprint Planning Meeting for Sprint 4. When more detail was needed during the actual development, the site in need requested the detailed information. A teleconference was arranged, and the help and information were made available. The implementing site asked the questions they needed clarification for, and wrote the specification themselves. Version 0.0 was available in one day, and the development could proceed. The specification was ready in four days. The site got exactly the paper they needed. It was not less, but also not more. It was at the time when they needed it, not sooner, but not later either.

Recommendation

Effective communication, collaboration, and coordination are the main contributing factors for success in Agile methods (Mishra and Mishra, 2009). A product development team developing a product requiring cross-discipline

engineering tries to capture the collective intelligence about the best way of turning market requirements into a product. The views of different stakeholders and each engineering discipline have dependencies, and continuous trade-off decisions are necessary during the development. The idea of pull information needs to be communicated clearly and early to developers. Encouragement and coaching are most likely necessary in the beginning. Asking for help, and therefore, admitting incapability, may be experienced as a sign of weakness in the existing culture. The bar for asking help should be lowered. You should also identify the people that can help in advance, so that you have a network of specialists ready for the developers.

Communication bridge

Description

After the initial communication problems, a person representing the engineering discipline from the other site was chosen for both teams to improve communication. This Communication Bridge practice was embraced by the teams, and was still being used in new projects at the time of writing this paper.

> *"Normally he only does the communication work between the mechanical designer, the hardware designer, and industrialization, and pcb designer. This person is only the bridge between the two sites ... Hardware people only speak the hardware language and mechanical people speak mechanical language. To communicate, they need to spend huge amounts of time to get understood."*
>
> *Scrum Master, China*

> *"We considered him the bridge. When we received a database from China and the mechanical specs ...then we made some changes and he translated those changes to China. He was more like a bridge. It was very good. I could feel that he worked very well with China, they actually took him as part of the team. They took his design suggestions. They took him seriously. So, it was good."*
>
> *Scrum Master, Mexico*

When studying inter-organizational product development, Paasivaara (2005) identified a practice called "communication through a resident engineer" in two case projects. In both cases, an engineer from the sub-contractor company worked closely with the customer company. The personnel in the first case project did not have enough practical experience to clearly say how valuable it is. The personnel in the second case project, however, were able to state benefits in several areas: speeding up the project, explaining the effect of

changes in design, finding contacts, lowering the bar for asking for help and translating the language between the two companies. Martin Fowler (2006) describes the practice of an Ambassador in his online essay "Using an Agile Software Process with Offshore Development." An Ambassador is a person from another country in an offshore setting helping teams to communicate on technical and business issues. The Communication Bridge practice in the case project resembles this, but the Ambassador is within the same country, only from a different engineering principle. Their role, however, is very similar to what Fowler describes. She helps by providing engineering discipline context. She fills small holes in information, which seem too insignificant for more formal communication between teams, and helps build trust between the teams.

Recommendation

A team must have all the necessary resources ready for use to achieve its goal. Developers from all the needed disciplines must be available for the team. If you have to do distributed development, you still have to solve this somehow. A solution can be an ambassador, a communication bridge or even distributed teams. A distributed team is an extreme variation of this communication practice. A case applying distributed teams is described in Sutherland, Viktorov and Blount (2006) and Sutherland et.al (2007). The teams were intentionally formed with members from two continents. The project had several teams, each divided between Utah and St. Petersburg. The objective was to solve the challenge of synchronizing work between sites in a distributed project setting. The results were excellent. The relative productivity of this large distributed project was almost the same as that of a small co-located Agile team used as a reference. The productivity was much higher than the industry average.

Technology

Description

Many technologies were tried in communication; blog, wiki, project document management system, product data management (PDM) system, Pro-Intralink, NetMeeting, Skype, and Acrobat 3D. Most of the tools that were officially supported by the corporate were experienced to be considerably complex or slow to use in daily collaboration. In addition to this, developers in Europe did not use the tools. Even technical files were transferred via email, which led to a version control nightmare.

> *"They [corporate tools] were slow. Nobody actually started to use them. They were lacking structures. Everyone was supposed to know how to use them, but in reality nobody knew. I felt, experience and training on tools was missing."*
>
> *Project Lead*

Despite the availability of different tools, email became the primary communication tool between the teams. This was mainly due to the time difference. Instant messaging did not have a major role, since the teams were not working at the same time. For this reason, teams developed a practice to use email effectively by keeping the discussion focused and response times short. A Scrum Master explained:

"It's kind of Daily Scrum but using emails."

Scrum Master, Mexico

Recommendation

The project team discovered a way of working as they proceeded. Similarly, tools should be evaluated continuously. Sticking with a tool which was mandated or decided in the beginning, but hinders the current progress, does not make sense. Defining rules on how to use these tools is also important. For example, simple matters such as naming conventions help a great deal in finding information, as was discovered in the whole project Retrospective meeting. Even when something is found out or generally considered to be bad, such as email, it can be tried again. In this project, email was found to be valuable because of the time difference. Emails were started again, after trying instant messaging, and the practice of efficient short emails emerged.

4.2.3 Improved commitment

Table 10. Summary of improved commitment.

Finding	Description	Recommendation
Collective product/project ownership	All team members took responsibility for and pride in the design and project. They felt equal in the project organization.	The whole team needs to be involved in activities regarding project, product and process. Explain the decisions the team is expected to make and why they are considered to be the best people to make those decisions.
Improvement through retrospectives	Several process improvements were made, mainly concerning communication, the prototyping process and the delivery of physical prototypes. The project lead was also active in removing the impediments outside the team's sphere of influence.	Enforce retrospectives to identify issues that are outside the team's sphere of influence. If possible, try to arrange cross-team retrospectives to improve the cooperation between teams. When the teams identify impediments, work promptly to remove them. Help from an external coach or facilitator can help in identifying the opportunities for improvement.

Collective product/project ownership

Description
Collective code ownership is one of the Extreme Programming practices. In other words, everyone in the team can change any part of the source code. In the context of the case project, this meant taking responsibility for the whole project as well as responsibility for one's own tasks. Team members were closely involved in planning, organizing, and managing the Sprints.

> *"… each and every member working in [] team has the possibility to give his opinion/ his time estimation for each single activity. The planning is much more realistic."*
>
> *Developer, Mexico*

The project lead asked developers openly for their opinions on the product and process. Everyone was treated as equal. Developers were encouraged to make design decisions themselves. When design conflicts occurred, they were not blamed, but supported in finding the solution together (see Failure is an option in chapter 4.2.1 and Trust in chapter 4.2.2). Presenting one's own work in the Sprint Review gave an opportunity to take pride in the achievement. Both Scrum Masters and team members emphasized this aspect. This in turn increased motivation and commitment. At first, this felt strange, but it eased very quickly. One Scrum Master quoted a developer in his team:

> *"She [team member] says it [the product] is my baby, I want to see it go all the way"*
>
> *Scrum Master, Mexico*

Involving developers this much in the process also changed the project managers' work. Both Scrum Masters (working as technical leads and project managers in earlier projects) were to some degree proponents of a coaching style of leadership. Nevertheless, the other Scrum Master said in the post-project interview that at the later stage of the project, with the time pressure, he felt powerless:

> *"I was not able to push the team…"*
>
> *Scrum Master, China*

In the whole team Retrospective, all leads and developers interviewed said that they would like to continue using Agile practices. This of course is one of the key measures of process change. If you cannot get people to believe and commit to chosen ways of working, it does not stand a chance.

Recommendation
You need to involve the whole team from the beginning. Only then will they have a chance to become a true, jelled, team (Katzenbach and Smith, 2003).

You have to explain why you want the team to take more responsibility over a broader area. Empower them by explaining the decisions they are expected to make, and why they are the best people to make them. You need to be empowered to be able to take responsibility. Offer your full support to help them learn these new skills. Do not expect this to happen overnight. The existing culture may not be very supportive about taking responsibility. People will make mistakes while learning new skills. Blaming them for these mistakes needs to be consciously avoided. Team members should feel completely safe to make decisions. Do not forget about the people who used to have the responsibility for decision-making. Explain that it is now their new responsibility to help other people to make the decisions.

Improvement through Retrospectives

Description

In the beginning, there was no detailed solution for how to apply Agile methods in the given context. During the initial introduction of the idea, we only shared our experience so far. We provided general knowledge about Agile development, and Scrum specifically. We moved into incremental planning, empowered the local teams, and enforced the Retrospective meetings to incrementally improve the process as well. Initially, Retrospectives had to be enforced to happen. Later, the teams understood their importance, and a pattern emerged that each team presented their results from their Retrospective in the Sprint Review meeting. They were again encouraged to also raise issues outside the team's sphere of influence. Several ideas were brought up, typically relating to lack of information or means of communication, the prototyping process or the distribution of physical prototypes. The project lead in Europe managed to improve these matters, or to remove obstacles hindering progress in most cases. On being asked why the team members got more engaged with the project, the project lead answered:

> *"I believe because we actively asked them. We kind of took them along. You can decide, you can tell what is working and what is not… We were on the same level with the developers, discussing what they thought about things. And we reacted to their feedback."*
>
> *Project Lead*

A cross-team Retrospective on the whole project was also conducted. The teams were able to identify several areas for improvement during this wider reflection. For example, they decided to look for local prototype suppliers to cut delivery times. Furthermore, they created a shared naming convention for technical files. The teams also decided to start using a higher abstraction level on Backlog items. Moreover, it was agreed that training on Agile development for stakeholders was needed. Overall, the improvements created during this

meeting focused more on issues outside the individual team such as cooperation between the teams or working with the stakeholders. Normal Sprint Retrospectives focused more on each team's local work.

Figure 14. Teams together in Project Retrospective.

Having an external facilitator available to answer questions and provide feedback during the project was seen as very valuable. Even for a simple framework, a two-day training course alone will not change the process of the whole project. The author attended many of meetings as an observer and was able to provide help on improvements. As a simple example, when the project was a few Sprints in, after observing the teams' meetings, the author suggested involving team members more in planning and reviewing. This is typical in Agile development, but without enforcing it, it may be forgotten.

Recommendation

The Scrum framework works as a catalyst for adapting practices. Continuous inspection and adaptation cycles guide the team to develop its practices iteratively as well. To help the team buy-in to the continuous improvement, you should work promptly to help them to address the identified problems. They should be encouraged to also point out problems outside their influence. In a multi-team and/or distributed project environment, you should invest time in Retrospectives on the project level, not only on the local team level. When improvement ideas are created, it is important to actually implement them. Takeuchi and Nonaka (1986) identify that the development department often works as a source of organizational change.

A full-time external coach would be very useful to remind people of the basic practices, but also enforce continuous improvement. An experienced process facilitator will enforce the Retrospective meetings and accelerate the learning process, even if they do not have the answers themselves. Having an external

coach to observe the process and team behavior can greatly help in identifying problems and possibilities for improvement.

4.2.4 Remaining and new challenges

Table 11. Summary of remaining and new challenges.

Finding	Description	Recommendation
Large-scale organizational change is needed	Project teams did not work in isolation, but had close interaction with the rest of the organization. Conflicts were identified in many areas, such as mind-set of Agile and other teams, resource allocation and the prototyping process.	Be prepared to change the existing processes. Schedule time to explain Agile values and principles to people outside the core team. Find a sponsor from the higher ranks to help in solving conflicts.
What is sufficient documentation?	There were no requirements for documentation in the definition of "done". Very limited written documentation resulted from the project, and this caused some difficulties later.	Include the level of sufficient documentation in the definition of "done," i.e. also plan to deliver documentation incrementally. Reserve some time at the end of the project for polishing the documentation for the future.
Need for a change in engineering practices	The overhead of practices became considerable, because of multiple design cycles. Investment in significant improvement within a single project engagement was difficult to justify.	Modern tools and technologies offer opportunities for growing automation. This can change design and testing to happen more simultaneously.

Large-scale organizational change is needed

Description

We experienced some conflicts at the boundaries of the project organization, for example management supporting multiple simultaneous projects, changing team members, the prototyping process, and collaboration with other functions.

We were not able to enforce a pure one-project environment and thus a constant team formation. Several developers mentioned this as negative during the first interviews through to the team Retrospectives. The developers themselves felt that they should be fully allocated to a single project. However, the resource management practices at the sites were relying on a traditional 100% utilization goal, and developers were from time to time allocated to different projects or to multiple projects simultaneously. Breaking this habit would require the education of management. Managers were willing to let this experiment continue and even supported it, but it did not affect existing processes and practices much. On the other hand, we were not active in enforcing and coaching them either.

> *"There was a general disbelief in my own working environment that we cannot just go and break the existing, trained, process model."*
>
> *Project Lead*

At the later stages of the case project, it also became evident that having other teams on the project working with a different mind-set is problematic. Technology development happened in a gray box aiming at big-bang delivery, and all the communication happened through the team leader. The Technology Development Site did not follow Agile project management. They were not synchronized at all with the incremental rhythm of the other two teams. Instead, they reported randomly, most often stating that they were almost ready. This difference in methods made it difficult to plan the incremental development.

> *"...at that time we were thinking that they are taking responsibility for the firmware and because they were not part of Scrum we had some difficulties understanding where they are going. At that time finally in [Case Project] one designer here in China [implemented] the firmware to make sure that the prototype works, because we cannot wait for the [European] designer to provide mature enough firmware to be integrated with the hardware."*
>
> *Scrum Master, China*

The official prototyping process at the sites was in contradiction to what was needed in this project. The process was developed to avoid ordering prototypes that would not work. In this project, the teams wanted to order prototypes to see whether the idea would work. Furthermore, rounds of these experiments were to happen frequently. This new focus on early prototypes instead of paper deliverables even created some tension at the sites:

> *"Peers certainly didn't like that so much. Prototypes were very early in the project, and they thought we were crazy and spending money on prototypes..."*
>
> *Scrum Master, Mexico*

Because the case project was not following the formal process, it was difficult for stakeholders to budget time for this project. The practice in official projects was for business to be heavily involved in up-front activities. Being involved throughout the project was new. Luckily, there were interested stakeholders to give feedback to the project. Further, the project had a vision of target cost, volume, and feature priority. This helped the project to stay focused, but more intensive feedback for navigating toward the goal was experienced as missing by the project lead.

Recommendation

At the beginning of an Agile pilot, it is likely that you have conflicts with the rest of the organization. This includes people in management positions. Traditional processes are developed to avoid mistakes, and to make it right the first time. In these processes, the reason for prototyping is validating instead of learning. Prototyping is a key practice in incremental hardware development, and prototypes are expected to be imperfect to maximize the learning. The process for prototyping needs to be extremely light and straightforward if it is to be done very frequently, as in this project, every four weeks. In Agile software development, the goal is a single button release, meaning that delivery of product can be done whenever without manual work. In hardware this would mean automating the process of ordering parts and formatting from a schematic, layout, or mechanics design tool to electrical delivery to the prototype suppliers. In addition, a partner-like relationship should be accomplished with your sub-contractor. This means that they understand the method you are using, and are willing to commit and collaborate to meet the required predictability. As said, this may totally contradict existing processes and practices, and these underlying assumptions create a huge change effort in themselves.

It is important to budget time for explaining to developers and other people outside the pilot project why the team is working differently. It is hard for an Agile team to commit to something that has dependencies outside the team, especially if there is no trust between the two. Therefore, the change will not last long if you do not engage others in the change process as well. There is a risk that they may feel it unfair that other teams are learning new things, while they have an obligation to follow the existing process.

You should try to find a Local Sponsor (first-line management support) and a Corporate Angel (high-rank executive support), as described by Manning and Rising (2005). If you can find both, your change management process is much easier. It pays to have a sponsor from a high enough rank that you could easily consult in case a change is needed. The project work is already hard and there is not usually adequate energy for driving these changes at the same time.

What is sufficient documentation?

Description

The official process in the case company was a document-oriented implementation of the Stage-Gate process. There was also a CMMI initiative, which focused on paper deliverables as well. In this project, the teams got

away from an up-front documentation obligation. The definition of "done"[4] did not include documentation. In the case project, teams were encouraged to pull information when they needed it. This information often remained in an informal format. There were exceptions, such as the documentation for the "push and push" concept (see chapter 4.2.2 for more on Pull information). Both Scrum Masters expressed in the post-project interview that they have needed more documentation after the project was finished.

> *"We don't care if it is Agile or [Waterfall], it's just a process, but they need to provide necessary documents for the project. Then they have to write the spec and this documentation time needs to be taken into account within the Sprint and the workload of the planning. This is a normal project, all integration project. Finally we need to deliver product and some documentation is necessary. Not just for the design team, but it is important also for other teams like industrialization and purchase, manufacturing and marketing. All the functions will need this kind of deliverable."*
>
> *Scrum Master, China*

> *"And that [limited amount of documentation] probably helped us to move forward very fast. But now you can say right away that you are missing a lot of information because you didn't record anything. Depending on your priorities, that might be a good thing or a bad thing."*
>
> *Scrum Master, Mexico*

Recommendation

Iterative work can easily be anchored to the Stage-Gate model, which would be a less dramatic change (Karlström and Runeson, 2006). However, the whole Agile philosophy is very different from the up-front documentation philosophy often associated with bureaucratic implementations of the Stage-Gate model. Nevertheless, a certain level of design documentation should be included in the definition of done. This does not necessarily mean written, well-polished, documentation, but for example, design notes included in actual drawings.

Agile software teams shift a large part of the traditional paper documentation work into running automated test cases and well-refactored clean source code. Some of these techniques might lend themselves to hardware development, but also some design documentation is needed (in software projects as well). Jim Highsmith (1999) explains how deliverables need to be monitored with the workstate instead of completeness over the project's timeline. In his

[4] The definition of "Done" is a mutually agreed list of criteria to be fulfilled before a work item can be considered complete, or "done" (Schwaber, 2004).

example, a deliverable goes through states: outline (conceptual), detail (model), reviewed (revised) and approved (available) state. Reserve the time to polish the documentation in the end to enable future work and also include this in your risk management plan in case of project cancelation.

Need for a change in engineering practices

Description
During this project, we identified a lack of appropriate technical practices in several areas. Many of them became visible due to the frequent prototyping. Frequent prototyping was experienced to cause extra effort. Materials for the prototype supplier needed to be gathered, and prototypes needed to be debugged after delivery. This is called the cycle cost of prototyping (see Figure 13).

> *"We were manufacturing both the hardware and mechanical parts. Sometimes it is a constraint to the project team. For me to make prototypes it is not like a one day job. For team it is a one-day job because they just need to make the files. […]*
> *Because what we can realize in four weeks is very limited and in the next four weeks we need to do it again and another mockup. To do a hardware or mechanical mockup is not so easy like firmware or software. Every time you make a hardware mockup, it takes time."*
>
> *Scrum Master, China*

> *"In my opinion, and this may be typical of physical products, hardware prototyping always demands effort. This effort is sometimes difficult to justify if it looks like the results will not be achieved. This prototyping problem was more evident the further we progressed. Toward the end the volumes of prototype series were growing and waiting times got longer due to the increased number of prototypes."*
>
> *Project Lead*

A single iteration was not able to deliver a full learning cycle. In the case project, iterations were often sequentially linked together as design, prototype and testing. This resembles the Waterfall, or sequential, development process.

Printed circuit board (PCB) and plastic designs were integrated virtually during the design, but conflicts were often uncovered when the actual physical prototypes were integrated.

> *"When we started to expect more mature products, we got problems. I believe it is due the fact that tools did not support this. We did not have 3D models for the components and we were not able to match the design to mechanics. PCBs were designed in 2D, without the 3D models, and design phase mistakes leaked into prototypes. Fixing them was for some reason very difficult."*
>
> *Project Lead*

Testing was involved as soon as prototypes were available, but it remained a manual effort. We recognized the need to increase automation, but we did not solve it. The main reason was that an out-of-the box solution was missing. Learning how to do it and then actually practicing it could be seen as investment, but the initial investment is difficult to justify in a single project scenario like this. Building experience on automation is a long-term investment decision.

> *"We have problems on the testing because we have a big delay on the burndown chart due the availability of the prototype or the prototype integration for the testing. We can see there will be one Sprint in June and even not significant progress during one month due to this kind of delay. That was not so good for the Sprint.[...]*
> *For example, firmware design takes two weeks, then we have to wait for the firmware to be ready for the whitebox testing, and this testing will take four weeks."*
>
> *Scrum Master, China*

Recommendation

While it of course is a good thing to find mistakes early in development, it is frustrating to continuously deal with issues that could have been easily noticed with proper tools. Teams looking for longer-term incremental hardware development should invest in tools capable of checking conflicts on the fly, with an extremely short feedback cycle. Agile software development teams apply a practice of continuous integration. After each change in the source code, the whole system gets built again, and an automated test suite is run to check that everything still works as it used to. Some CAD tools provide little such help. You can, for example, integrate 3D PCB layout design with your 3D mechanics design, but there is a lot to hope for. 3D printers, rapid PCB prototyping tools and for example, use of FPGAs could take integration frequency to a new level.

Testing systems like TI TestStand or Saab's TestManager, flying probe and boundary-scan techniques can be used to automate testing in the development phase of the life cycle, but they need investment. The investment is difficult or impossible to justify in a single-case off-shoring situation. Nevertheless, testing is one of the key matters that needs to be solved in incremental

hardware development. Jim Highsmith mentioned in a panel discussion at the Agile 2007 conference in Washington D.C. that he has coached a hardware team into Extreme Programming techniques, and this was particularly targeted at testing strategies (Highsmith, 2007).

5 Research design – systematic literature review

This chapter presents the research design for the systematic literature review. First, it gives an overview of the research approach and then a more detailed description of database search strategy, primary study selection and method to create a secondary study.

5.1 Research method

The research questions were introduced in chapter 1.2. The objective of the literature review is to summarize the existing research on knowledge transfer from Agile software development to the hardware development domain. "A systematic literature review is a means of identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area, or phenomenon of interest. Individual studies contributing to a systematic review are called primary studies; a systematic review is a form a secondary study" (Kitchenham, 2007).

There are many reasons for undertaking a systematic literature review. The most common reasons according to Kitchenham are:

- To summarize the existing evidence concerning a treatment or technology
- To identify any gaps in current research in order to suggest areas for further investigation
- To provide a framework/background in order to appropriately position new research activities

When we consider knowledge transfer from the Agile software development domain to the hardware development domain, it is fair to say that all of the above reasons are valid. The objective of the review is to collect and synthesize the current knowledge on applying learning from Agile software development into non-software, more generic, new product development. As presented in chapters 1.1 and 2.2, iterative and experimental models have been suggested for hardware development, but nevertheless, it has been the Agile software development models that are far more widely adopted. Despite the fact that knowledge about similar approaches to hardware development exists, this study focused solely on transforming knowledge about Agile methods from the software domain to the hardware domain.

Brereton and colleagues (2007) define a 3-phase, 10-stage, literature review process, illustrated in Figure 15. The process describes how to carry out a systematic literature review through planning, conducting and documenting the review.

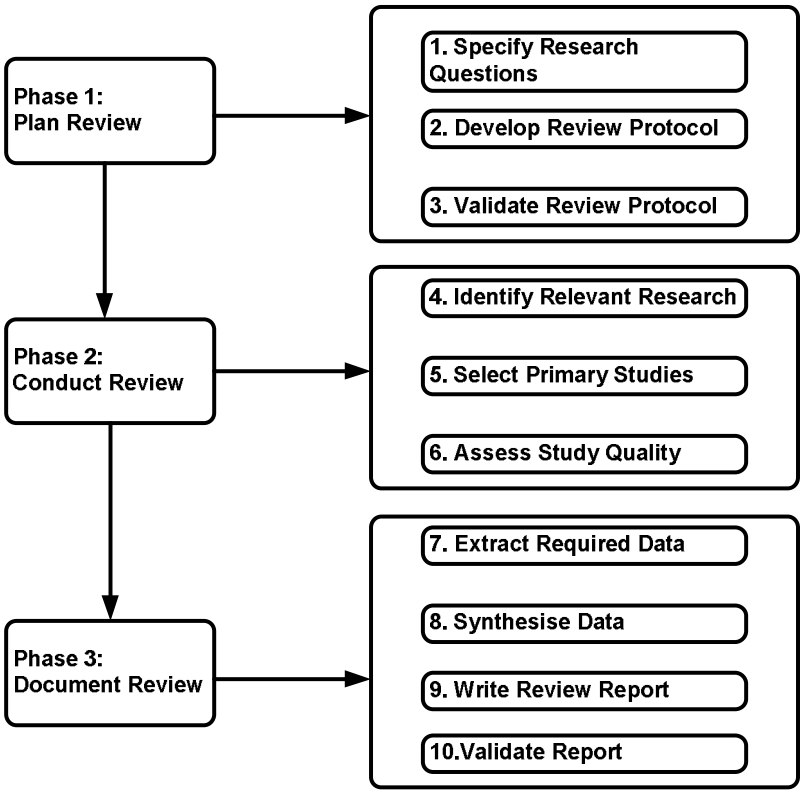

Figure 15. A 3-phase, 10-stage, review process (Brereton et al, 2007).

5.2 Database search strategy

The review was started by identifying keywords based on the research questions. After evaluating the results from quick runs on just a few databases, the keywords were adjusted and the search was run again. Adjustments were made by adding and removing synonyms and adding and removing terms. This takes some time as you want to get as good balance as possible between the total number of samples and including all relevant studies. The selected terms in two categories are presented in Table 12. Category 1 includes terms referring to Agile development. Terms in category 2 introduce the aspect of non-software development. Category 3 was also considered, but based on the test runs its use would have resulted in too narrow results. All pairs from category 1 and 2 were used to run the search. When possible, the search criteria were combined into single search using the Boolean “and” and “or” operators.

Table 12. Search terms by categories.

Category 1		**Category 2**		**Category 3 (Not used)**
OR	AND	OR	AND	OR
scrum		hardware		development
agile		hw		design
dSDM		electronics		build
“extreme programming”		mechanics		engineering
Xp		embedded		project
		“product development”		program
		“project management”		process
		non-software		method
		non software		practice
		non sw		
incremental		non-sw		
iterative				
flexible				
lid				
temporal pacing				
time-slotted		non it		
timeboxed		non-it		
time-boxed				

Database selection was based on ease of search, exporting capabilities and the results of few basic searches, for example “agile AND hardware.” The following databases were excluded from this review based on initial results:

- ACM Digital library (www.portal.acm.org/dl.cfm)
- ISI Web of Science (http://apps.isiknowledge.com/WOS_GeneralSearch_input.do?product=WOS&search_mode=GeneralSearch&SID=N1Hl8IjCEL@dOh6f7EO&preferencesSaved=&highlighted_tab=WOS)
- Google Scholar (scholar.google.com.au/)
- AIS eLibrary (http://aisel.aisnet.org/)

Table 13 summarizes the results from selected electronic databases. The used search criteria, specific issues and initial number of search hits are listed for each database.

Table 13. Results from search run[5].

Database	Search Criteria	Specific	Hits
IEEE Xplore http://www.ieeexplore.ieee.org	(scrum OR agile OR dsdm OR "extreme programming" OR xp OR incremental OR iterative) AND(hardware OR hw OR mechanics OR electronics OR embedded OR "project management" OR "product development" OR "non-software" OR "non-sw" or "non software" or "non sw")	Selected; - 'metadata only' (i.e. title, abstract and keywords)	719
Elsevier ScienceDirect http://www.sciencedirect.com	TITLE-ABSTR-KEY(scrum OR agile OR dsdm OR {extreme programming} OR xp) AND TITLE-ABSTR-KEY(hardware OR hw OR mechanics OR electronics OR embedded OR {project management} OR {product development} OR {non-software} OR {non-sw} OR {non software} OR {non sw})	Selected; - Journals only, - Subjects Computer Science and Engineering, - 1990 - present	83
Compendex EI http://www.engineeringvillage2.org/	((scrum OR agile OR dsdm OR {extreme programming} OR xp) AND (hardware OR hw OR mechanics OR electronics OR embedded OR {project management} OR {product development} OR {non-software} OR {non-sw} OR {non software} OR {non sw})) wn KY	Selected; - Compendex database only. - Only records in English. - 1990 - present Documents were difficult to retrieve.	1301
Scopus http://www.scopus.com	TITLE-ABS-KEY(agile or scrum OR "extreme programming" OR dsdm OR xp) AND TITLE-ABS-KEY(hardware OR mechanics OR electronics OR embedded OR "product development" OR "project management" OR "non-software" OR "non-sw" OR "non software" OR "non sw") AND PUBYEAR AFT 1989	Service was down and search was run 2 weeks later than others.	1571

[5] Search was run on 7.3.2010 except for Scopus.

SpringerLink http://www.springerlink.com/	abstract:((scrum OR agile OR dsdm) AND (hardware OR hw OR mechanics OR electronics OR embedded)) abstract:(({extreme programming} OR xp) AND (hardware OR hw OR mechanics OR electronics OR embedded)) abstract:((scrum OR agile OR dsdm) AND ({non-software} OR {non-sw} or {non software} or {non sw})) abstract:(({extreme programming} OR xp) AND ({non-software} OR {non-sw} or {non software} or {non sw}))	Only accepts 10 search terms. Needed to run in 4 batches (hits: 58, 23, 0, 0)	81
Wiley InterScience http://www3.interscience.wiley.com	All pairs of category 1 and category 2 search terms with AND operator. Exceptions are ‘xp’ and ‘hw’. They were identified as ‘too generic’ by the database search engine.	Selected; - 1990-2010 Search criteria resulted in high number of duplicates.	437

5.3 Primary study selection

After the initial database search had been done, the results needed to be analyzed for relevancy. Figure 16 illustrates the stages and techniques for this analysis. The initial search found 4192 hits. Duplicates were removed using EndNote[6] software, but also going through the reference list manually. This resulted in 2358 hits, although during stage 2 some missed duplications were further identified and removed from the sample. The large number of duplicates can be explained by having to use separate searches in the Springerlink and Wiley InterScience databases. During stage 2, all titles of studies were evaluated and those clearly outside the scope of research questions were removed from the sample. Only 503 studies passed this stage. A few common issues contributed to the large number of irrelevant studies at this stage. The term "Agile" itself forms a problematic search term. As an example, many hits described the development of a device and one of its attributes was described to be agile, e.g. agile navigation or agile motion. The "XP" abbreviation for Extreme Programming is another problematic term, because it returns many references to Windows XP. The term "scrum" resulted in sports medical publications in some databases without narrowing the journals included, because it is a rugby term. If the title did not give many hints about the content, the reference was included in the next stage. During stage 3, it was time to go through all the abstracts and keywords in more detail. The evaluation focused on the study's relevance to the research question. If in doubt, the study was included in the next stage. Books and everything not written in English, were excluded. There were 156 studies left when moving to stage 4, the final relevancy screening.

A full copy was retrieved for all 156 studies left from stage 3. Stage 4 involved going through the full paper and selecting primary studies using the screening question as criteria. The earlier stages had already revealed that the number of quality studies would be low. For this reason, the final screening was based on a single question to make sure the study had implications for the research question:

> *Does the study present data, empirical or theoretical, on the applicability of Agile Methods, or practices and techniques associated with them, to other engineering disciplines than software?*

[6] http://www.endnote.com/

Studies based on the same data were excluded according to the guidelines by Kitchenham (2007). Ten primary studies were selected at the end of stage 4. References from selected primary studies were analyzed, but relevant additional studies were not found. The 10 primary studies are listed in Table 14 in chapter 6.1.

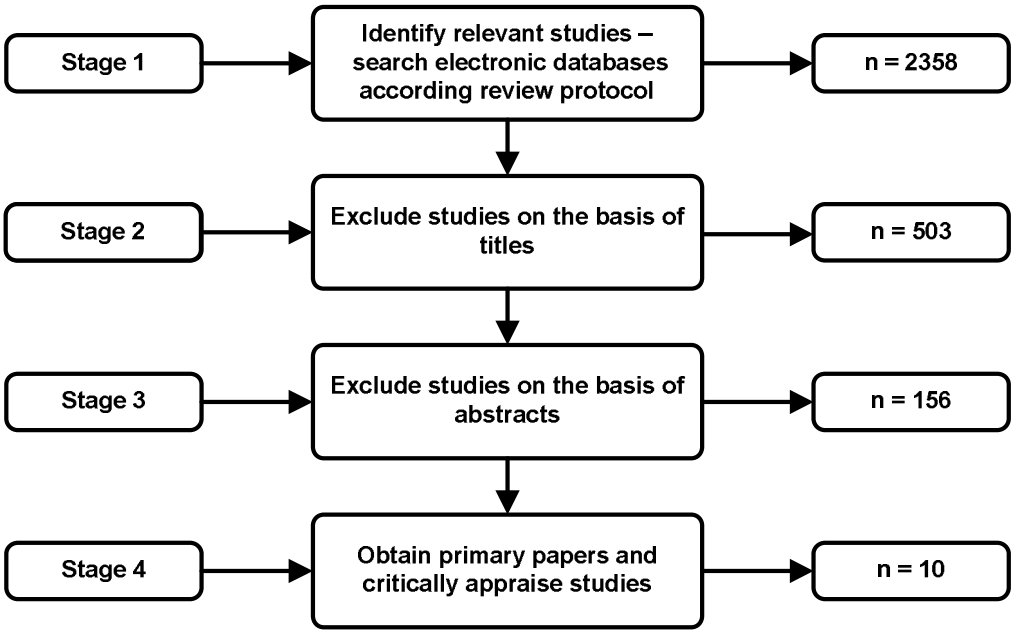

Figure 16. Process of selecting primary studies.

5.4 Creating a secondary study

Synthesis was conducted as a line of argument, as described by Noblit and Hare (1988). This approach is used when researchers are concerned about what they can infer about a topic as a whole from a set of selective studies that look at part of the issue. The analysis can be considered to have two phases. First, the individual studies are analyzed to find out the similarities and differences. Then, a grounded theory is developed to interpret the "whole". Issues of importance are identified and the approach to each issue taken by each study is documented and tabulated (Kitchenham, 2007).

The explanation of the synthesizing can easily lead to an understanding that it is the last phase of a sequential process, but it is far from that. Analyzing studies together on how they relate, and how to understand the whole, is a very iterative process. It is impossible to separate the phases of the process. The synthesizing started at the very outset of the review.

First, the relevant data from primary studies needs to be extracted. Ideally, data from studies is extracted using a standardized form. This was a challenge,

as the studies varied widely in their style and scope. When the studies were first read, only basic data was extracted, such as the type of study, the Agile method, engineering discipline, positive and negative findings, what supported the use of Agile methods, and other generic observations. These categories were just a starting point. There were several rounds of reworking the categories. Each time, the new version was tested by going through the studies again. The result of this iterative process is presented in APPENDIX A in table format. The rows in the table represent the data categories. Individual studies are listed in the columns. Short direct quotations with a page number were used to record the relevant data from primary studies. The quotation is recorded in the corresponding cell in the table. If the study does not address the data category, the cell is left empty. This makes finding the original context easy when doing the analysis. The second last row presents any quotes that felt important, but did not fit any of the main categories. The last row contains additional interesting observations which are not direct quotes. In the end, the following main themes emerged from the data categories:

- Co-design
- Testing
- Iterative hardware development

Each theme was first interpreted individually to synthesize the data from different studies. The first theme covered the main cause for the need for hardware teams to start looking at Agile development. When embedded software developers start using Agile methods, it affects the hardware development as well. The next two themes were related more to implementation from a practical perspective. Testing activities are moved forward in Agile software development, compared to traditional validation at the end. The same applies to non-software development. Finally, iterative hardware development was identified to be not just possible, but beneficial. Secondary interpretation was created by asking how the themes are linked together and what they infer about the whole. This revealed an enforcing cycle between co-design, testing, and iterative hardware development.

It was evident that the primary studies supported each other. This was expected, as experience reports of negative findings or failures are very uncommon. While this is a limitation, it also means that we can build a line of argument based on primary studies. The next chapter presents the results of this analysis in more detail.

6 Results – Systematic literature review

This chapter presents the results from the systematic literature review. First, an overview lists the selected primary studies and presents their key parameters. The second sub-chapter provides the synthesis and key findings from the review.

6.1 Overview

Through the review process presented in the previous chapter, ten primary studies were selected. Table 14 lists the studies. The ID in the table is used later in this chapter when referencing the primary studies.

Table 14. Primary studies.

ID	Reference
1	(Allen, Abdel-Aty-Zohdy and Ewing, 2009) Allen, Jacob N., Abdel-Aty-Zohdy, Hoda S. Dr., Ewing, Robert L. Dr., Agile Hardware Development with Rapid Hardware Definition Language, IEEE, 2009.
2	(Boehm, 2006) Boehm, Barry, Some Future Trends and Implications for Systems and Software Engineering Processes, Systems Engineering, Vol.9, No.1, Wiley Periodicals, Inc, 2006.
3	(Chae et.al., 2006) Chae, Heeseo, Lee, Dong-hyun, Park, Jiyong and Peter, Hoh, The Partitioning Methodology in Hardware/Software Co-design Using Extreme Programming: Evaluation through the Lego Robot Project, Proceedings of The Sixth IEEE International Conference on Computer and Information Technology (CIT'06), IEEE, 2006.
4	(Cordeiro et.al.,2007) Cordeiro, Lucas, Barreto, Raimundo, Barcelos, Rafael, Oliveira, Meuse, Lucena, Vincente and Maciel, Paulo, Agile Development for Embedded Systems: A Platform-Based Design Approach, Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'+7), IEEE, 2007.
5	(Doran, 2003) Doran, Hans Dermot, XP: Good for Anything Other than Software Development, XP 2003, Springer-Verlag Berlin Heidelberg, 2003.
6	(Kettunen and Laanti, 2008) Kettunen, Petri and Laanti, Maarit, Combining Agile Software Projects and Large-scale Organizational Agility, Software Process Improvement and Practice, No.13, 183-193, 2008.
7	(Paelke and Nebe, 2008) Paelke, Volker and Nebe, Karsten, Integrating Agile Methods for Mixed Reality Design Space Exploration, DIS'08, 2008.
8	(Van Schooenderwoert and Morsicato, 2004) Van Schooenderwoert, Nancy and Morsicato, Ron, Taming the Embedded Tiger – Agile Test Techniques for Embedded Software, Agile Development Conference, 2004.
9	(Smith, 2008) Smith, Preston G., Change: Embrace It, Don't Deny It, Research-Technology Management, Vol.51, No.4, 34-40, 2008.
10	(Suhaib, Mathaikutty and Shukla, 2004) Suhaib, Syed, Mathaikutty, Deepak and Shukla, Sandeep, Extreme Formal Modeling (XFM) for Hardware Models, Proceedings of fifth International Workshop on Microprocessor Test and Verification (MTV'04), IEEE, 2004.

Table 15 shows the distribution of the publishing year of primary studies. We can see that the amount of literature stayed low throughout the decade, and no trends were found based on this review.

Table 15. Distribution of studies by year of publication.

	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010
Papers	0	0	1	2	0	2	1	3	1	0
Percentage	0%	0%	10%	20%	0%	20%	10%	30%	10%	0%

Table 16 summarizes the key parameters of the selected studies. If the study focused on more than one sub-area, such as Scrum and XP or hardware and co-design, it counts for both. First, the type of study is considered. All the primary studies are either lessons learned type experience reports or technical papers. This leaves no evidence from systematic academic research. Second, the studies are categorized by the specific Agile method they reference. Half of the studies (50%) reference Extreme Programming, which defines programming practices. This was surprising, since Scrum is the most commonly adopted Agile method in the industry. Furthermore, Scrum focuses on project management practices, making it easier to adopt outside software development. Scrum was mentioned only in 20% of the studies. Third, studies were evaluated based on the engineering discipline (if any) they focus on. Most studies (70%, 7 out of 10) discuss co-design between different engineering disciplines. This can be explained by the fact that this review focused on using knowledge from Agile software development in other disciplines. The search terms were selected from the software discipline and that most likely affected the result. Despite this, 40% of the studies discuss hardware development.

Table 16. Summary of key parameters of the primary studies.

	Number of papers	**Percentage**	**References**
Type of study			
Experience report/ Lessons learned	5	50%	[3, 5, 7, 8, 10]
Technical paper	5	50%	[1, 2, 4, 6, 9]
Agile Method			
XP	5	50%	[3 - 5, 8, 10]
Generic	4	40%	[1, 2, 6, 9]
Scrum	2	20%	[4, 7]
Engineering Discipline			
Co-design	7	70%	[2-8]
Hardware	4	40%	[1, 7, 8, 10]
Generic NPD	1	10%	[9]
Embedded SW	1	10%	[8]

6.2 Key findings

The review was focusing on finding out how the knowledge from Agile software methods has been transferred to the non-software domain and the implications of this transfer. During the review, three themes emerged: co-design, testing, and iterative hardware development. In addition, the secondary interpretation revealed an enforcing cycle between them. The synthesis of the data is summarized in Table 17.

Table 17. Synthesis of data from primary studies.

Theme	Primary interpretation	Secondary interpretation
Co-Design	Extended collaborative co-design with cross-discipline up-front prototyping has helped teams to achieve several benefits, such as efficiency, innovativeness and system-level optimization.	In systems development, the challenges that Agile development tackles are shared between engineering disciplines. When an increasing number of embedded software teams adopt an Agile process, it creates a need for hardware development to change its way of working.
Testing	Early and frequent prototyping used in iterative hardware development puts testing forward into the process. This helps in detecting mistakes earlier, but it also introduces more testing work. Testing practices need to be improved and developed toward more automation.	Extended co-design results in increased innovation, not just in the final product, but also in development practices. For example, by utilizing the latest technology together with software and hardware developers, novel automated testing practices can be created.
Iterative Hardware Development	Iterative development is identified to suit hardware development, addressing challenges such as learning and change. However, hardware development lacks the flexibility of software development, and a need for developing new types of maturing prototypes and more partner-like relationships with suppliers is recognized.	Innovation in project and test automation reduces the overall cycle cost. This makes iterative hardware development more attractive, which in turn brings software and hardware development closer together. Therefore, co-design, testing and iterative hardware development using up-front prototyping creates an enforcing cycle.

6.2.1 Co-design

Almost all of the studies ([2-8]) present the need for extended or continuous collaboration between different engineering disciplines, and identify it to bring several benefits, such as efficiency, innovativeness, and system-level optimization.

An interesting perspective on Agile adoption is presented in the domain of mixed reality by Paelke and Nebe (2008). They present a case on the development of an augmented paper map (an electronic device providing additional information to a paper map). The resulting design was significantly different from existing solutions. They suggest that cross-discipline exploring

and analysis contributed to this result. Mixed reality development involves a large amount of bleeding edge technology and novel user experience concepts. This sets up an interesting challenge, as all the involved disciplines are creating something new, and up-front partitioning and management with traditional methods is difficult or even impossible. For this reason, they suggest that hardware, software, and user experience design must be equally considered and that Agile methods using short iterations and rapid feedback on prototypes is the better match for such an exploratory process. Prototyping can proceed from lightweight prototypes, such as design sketches, paper prototypes, and mock-ups, to partial implementation, and eventually full prototypes. Coordination between the disciplines is easier when everyone uses the same iterative Scrum-based framework. Furthermore, Doran (2003) claims that when using XP in a co-design project, they achieved an unusually high number of interactive features and elements in their design in a short time.

Similar, but more generic ideas are presented by Kettunen and Laanti (2008). They emphasize that embedded software development is not done in isolation. It has many dependencies, both internal and external to organizations. Agile affects the whole organization, not just the software development. If an embedded software team starts using Agile methods, this has implications for other disciplines as well. Hardware development is an obvious example. Kettunen and Laanti propose development process adaptations to help an organization become more flexible, such as flexible product architecture based on standardized hardware/software interfaces, close cooperation between software and hardware, and continuous iterative integration. A system-wide approach to Agility is needed in NPD organizations.

Other authors continue to emphasize the need for a collaborative co-design phase. Chae and his team (2006) explain how an extended co-design period helps in reducing the cost of mistakes by making it possible to detect them earlier. Furthermore, software and hardware disciplines working together on automated and semi-automated testing has been identified to lead to a different relationship between software and hardware teams (Van Schooenderwoert and Morsicato, 2004).

One often-heard argument against the use of Agile methods in embedded system development is the presence of special characteristics, such as energy consumption, execution time, and memory constraints. Cordeiro and his colleagues (2007) turn this upside down and propose a methodology based on XP and Scrum just because of these characteristics. They list changes to methodology they made to make Agile development applicable. The changes were: “(i) adopt processes and tools to optimize the product’s design rather than take paths that lead to designs that have no chance of satisfying the constraints, (ii) support software and hardware development through a comprehensive flow from specification to implementation, (iii) instantiate the

system platform based on the application constraints rather than over-design a platform instance for a given product, and (iv) use system platform to conduct various design space exploration analyses for performance." To accommodate these changes, they further defined Scrum's Product Owner as divided into three roles: 1) Platform owner 2) Product leader and 3) Feature leader. These roles are introduced as the complexity of the project grows. Unfortunately, the proposed method is only evaluated on a theoretical level.

Quantitative data is rare in the primary studies. The paper on co-design practice at early phases of embedded system development by Chae et al. (2006) is a welcome exception. They describe an attempt to transfer the ideas of Extreme Programming to co-design. The method is called PAMUX (Partitioning Methodology Using Extreme Programming). It proposes incremental hw/sw partitioning and continuous testing of the partitioning through integration. Furthermore, software development is started before the hardware is available. This is made possible by using unit testing and stubbing the hardware interface. The process was applied to a university class case study co-developing an application of a Lego robot. They compare four teams, two using a traditional co-design process and two using an iterative process with continuous integration of hardware and software. The results were clearly positive for the use of the iterative model measured in efficiency (in terms of lines of code), but more importantly measured in calendar time. The calendar time was cut in half using iterative development. As this was in a university setting the real customer satisfaction aspect remained unexplored.

Boehm (2006) forecasts trends in systems development and the likely influences of these trends on development processes. He identifies eight trends: the increasing integration of software engineering and systems engineering, an increased emphasis on users and end value, increasing SIS (software intensive system) criticality and need for dependability, increasingly rapid change, increasing SIS globalization and need for interoperability, increasingly complex system of systems, increasing needs for COTS, reuse, and legacy SIS integration and computational plenty. He emphasizes the importance of integration and concurrency of systems and software engineering processes, including hardware development and people processes. Boehm proposes a balance between more emergent Agile and plan-driven parts of the project. These are supplemented with concurrent validation and verification processes.

6.2.2 Testing

Testing is addressed by several studies ([1, 3, 5, 8, 10]). The importance and cost benefits of moving testing forward are identified and agreed on. Test automation opportunities differ according to the system being developed. For example, possibilities are more versatile for ASIC/FPGA development than for

a design also requiring electronics and mechanics development. Nevertheless, the need for automation is clear and some advice on techniques is presented.

The paper by Van Schooenderwoert and Morsicato (2004) is about embedded software testing techniques, but discusses the implications for hardware development. The sw/hw integration testing was helped with a hardware unit test practice. Hardware unit tests eliminated the need to run the whole system when testing the software/hardware boundary. The basic correctness of the software was tested continuously. This provides at least a partial answer to the challenge Doran (2003) reports: "…in the area of system design, it did not appear to occur to anybody that one could determine how a system is to be tested before it is actually built." This practice also made it easier to isolate the cause of defects. Chae et al. (2006) remind about the need for manual testing to cover system-level testing, which is not covered by XP practices.

Suhaib, Mathaikutty and Shukla (2004) document a processor development process based on Extreme Programming practices. Practices such as user stories and test-driven development are explained in more detail. An example of a user story in this domain is "each instruction executes in a certain order." A description in linear time property can be defined for stories written in this style. Test-driven development can be adapted by defining the time properties for each story in advance. The method is tested in three cases: the DLX pipeline, monitoring of the ISA bus and the arbitration phase of the Pentium Pro bus. Based on the results of the experiments, several benefits are identified, such as avoiding implementing extra features and identifying mistakes immediately. Allen, Abdel-Aty-Zohdy and Ewing (2009) present an experiment in processor development. They developed a framework for designing FPGA hardware using .NET languages such as C#, F# or Ruby. The framework is called Rapid HDL and is designed to move FPGA development toward Agile software development. By using a framework like Rapid HDL, test-driven development techniques can be brought into hardware development.

6.2.3 Iterative hardware development

Many of the papers focus on co-design and not on hardware-specific issues in detail. Most of the studies still address iterative hardware development ([1, 2, 5-10]). The data gives evidence of the applicability of iterative development to hardware development.

Hardware is always developed in increments, in the experience of Van Schooenderwoert and Morsicato (2004). The design often matures from evaluation boards into integrated production quality design. They turn the question of whether Agile methods are applicable to hardware development upside down: "In embedded development, the hardware is always changing

…Any embedded software development strategy must deal with changing hardware."

Doran (2003) reports results from a few initiatives on applying Extreme Programming practices to hardware development. He identified improvement in interaction with the customer. Nevertheless, Doran claims failure due to the prototyping cost and long lead time for the prototype sub-contractor. This means that when hardware development is following fast-paced development, it also affects sub-contractors. A more partner-like relationship is needed to achieve trust in delivery times. Another barrier was the incapability to solve the testing challenge.

The study by Suhaib, Mathaikutty and Shukla (2004) explains how the state space grew incrementally in one processor development project. This can be interpreted as evidence of a possibility of incrementally developing such a design. As mentioned earlier, Paelke and Nebe (2008) add data from incremental development requiring effort from multiple engineering disciplines. They adopt the idea of an incrementally growing product with maturing prototypes which frequently integrate the effort from different disciplines. Boehm (2006) acknowledges that software development often advances several increments between major hardware increments. However, organizations that have synchronized these increments have gained an advantage.

The technical paper by Preston Smith (2008) references Agile software development and suggests that the principles can be applied to development outside software as well. Smith lists nine tools and practices that provide more flexibility for product development:

- Continually monitor customers
- Fence-in change
- Try things out
- Explore the design space
- Build strong teams
- Make decisions at the last responsible moment
- Plan piecemeal and constantly consider risk
- Maintain flexibility in upper layers of process
- Out-innovate the competition

The previously mentioned system-wide flexibility (Kettunen and Laanti, 2008) supports this thinking. Throughout the paper, Smith reminds that flexibility in product development comes with a cost. Benefit-cost analysis needs to be done to identify the areas where flexibility techniques such as modular architecture with interfaces provide the greatest benefit. The variables in this analysis have changed dramatically in recent years due to advances in technology. New development technology, such as 3D printers, continues to reduce the difference between hardware and software development. The diminishing

differences between software and hardware development can also be concluded from the description of a framework to develop processors in any .Net language by Allen and his team (2007).

6.2.4 Secondary interpretation: Enforcing cycle

Agile software development is popular because it provides help with modern-day development challenges, such as a faster time-to-market and rapid rate of change. In systems development, these challenges are shared by multiple engineering disciplines. Embedded software teams are moving into using Agile software development, and this changes how embedded system-level development is done. Agile embedded software developers want to test their design concepts in real hardware as soon as possible. Product Owners and Agile managers want to see something functional at a frequent cadence. This means that system-level integration is requested frequently. All this leads to a common approach toward fast-paced experimenting.

When software and hardware developers work more collaboratively during the whole project, they will find ways to move the testing forward in their process. Furthermore, technology advancement changes the game from both ends. Technology that goes into products gets more naturally flexible (FPGA), and technology that is used to develop reduces the cost of experimentation and change (3D printers, CAD technology and rapid prototyping). Technology is making faster hardware cycles possible. By using the whole team approach, you accelerate the speed of adapting these new technologies. New opportunities for improving testing and automation will be revealed. Up-front testing becomes more efficient. The belief that testing partial and non-functioning prototypes is expensive, or unnecessary rework, is no longer valid.

Iterative hardware development introduces a cycle cost of prototyping, but this cost can be lowered by the above-mentioned innovations in practices. However, other improvements are possible as well. For example, having a different, partner-like, relationship with prototype suppliers can dramatically shorten the lead time for acquiring physical prototypes. These and other changes will reduce the resistance, and iterations for hardware development can be shortened. The difference between Agile embedded software and hardware development processes diminish. This can be called Agile co-design.

By doing this, all disciplines can achieve an even faster experimentation rhythm. This will increase the amount of innovations in the system. All this together forms an enforcing cycle in Agile co-design (Figure 17).

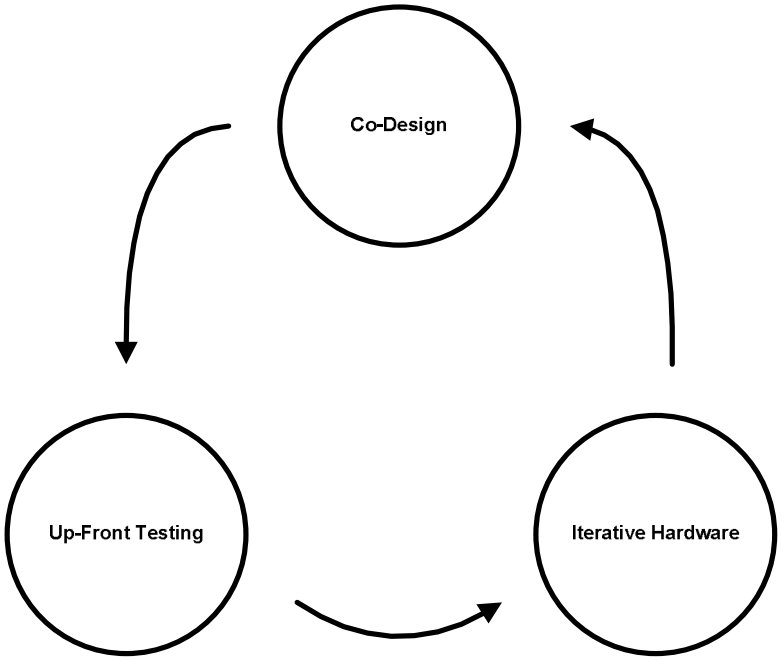

Figure 17. Enforcing cycle in Agile co-design.

7 Discussion

This research consisted of a case study and a systematic literature review on knowledge transfer from Agile software development to the hardware development domain. The case study provided knowledge from an industrial setting. Afterwards, the systematic literature review was conducted to obtain a wider angle on existing knowledge. The results from the case study and the systematic literature review were presented in chapters 4 and 6 respectively. This chapter takes a look at both studies together. The first two sub-chapters give a brief summary of the studies and a comparison of their findings. The next sub-chapter provides further discussion on organizational implications. The following sub-chapter answers the research questions. The final sub-chapter considers the limitations of the study.

7.1 Summary of the case and literature studies

The case project was the distributed development of a networked mains control system. The project organization began following Scrum, which is an Agile project management framework. Scrum gives guidance for project management, and also includes elements for iterative process improvement. Process improvement was done throughout the case project. The case study's findings were grouped into four areas: accelerated learning, improved communication, improved commitment, and remaining challenges.

Accelerated learning was a result of short experiments in the case study (see emerging product, Figure 9 in chapter 4.1.2). This was vitally important, as a large amount of knowledge needed to be transferred to the development teams. Actually, the need for the transfer of knowledge itself had to be learned at the outset of the project. Fast-paced iterative development, focusing on working design, revealed gaps in knowledge and forced the project organization to find a solution. It was evident that the lack of knowledge would hinder the speed of development. The project team tackled this by arranging a domain expert to be reachable by the team. They further agreed that if the domain knowledge was not available, the team was encouraged to make design decisions according to their own best understanding. Mistakes would be corrected based on feedback on future iterations.

Several factors contributed to *improved communication*. The distributed characteristic of the case project is outside the scope of this research, but it affected the challenge in communications. The geographical distribution emphasized that the teams were divided by engineering discipline. The case study demonstrated that it is difficult to have a successful distributed project without intensive communication. Team distribution can be geographical or by engineering discipline, or as in the case project, both. Independent teams can

work with specifications, but they will face difficulties when the results are integrated only at the end. While not easy either, it is easier to solve the daily communication challenges. In this project, the fundamental discovery was the need for a communication bridge, a local representative of the other team's engineering discipline (see team communication mechanisms, Figure 10 in chapter 4.1.3). This helped the teams to understand the requirements and constraints of the other discipline. Sosa et al. (2002) list practices to overcome difficulties caused by distances between teams, such as a high degree of team member interdependence, strong organizational bonds, and the use of electronic communication media. This study supports this and implies that they are listed in prioritized order. This project did have help from tools, but with improper training and limited experience, they would have been left unused without the strong bond and continuous reflection on working habits between the teams.

Empowerment and shared responsibility for the project as a whole resulted in *higher commitment and motivation*. Everyone in the project was involved in project activities, such as Sprint Planning and Review meetings. Teams were also empowered to improve their own process through the Retrospective meetings. This increase in the level of engagement made people identify themselves with the whole project. They were proud to be part of it.

Remaining and new challenges were also identified: large-scale organizational change, the amount of documentation and engineering skills. Agile development teams do not work in isolation. The development function has strong dependencies on other functions and external organizations. Others need to adapt their way of working to accommodate rapid incremental development. Existing processes may be very different, and even grounded within a completely different value system. An example in the case study was the existing prototyping process designed for avoiding mistakes. It completely contradicted the approach under experiment.

Prototyping also made the lack of appropriate technical practices visible. When several rounds of prototypes need to be tested, manual testing becomes quite laborious. In the case project the idea of automated testing, or test-first development, was explored, but the teams did not get to experiment with this in practice. Effective testing is one of the key problems to be solved. With modern FPGAs, circuit board prototyping, in-circuit testing, boundary-scan technologies and 3D printers, we can reduce the cycle cost and make simple partial prototyping more attractive: the cycle time can be reduced to hours. Of course this definition of prototype is quite different from the more traditional product development literature. In order to fully exploit this, the testing practices need to evolve as well.

A systematic literature review was conducted to find out what is currently known about knowledge transfer from Agile software development to new

product development in general. A protocol was developed to conduct the review. The first stage of the protocol was a database search. The search resulted in 2358 studies matching the set of keywords. Next, the studies were analyzed for relevance following the review protocol. During this stage, it became obvious that there are only a limited number of studies on the subject. Ten primary studies were selected for analysis. Despite the low number of primary studies, the synthesis was able to reveal three themes: co-design, testing, and iterative hardware development. Furthermore, secondary interpretation during the data synthesis identified an enforcing cycle between the three themes.

The primary studies mainly focused on *co-design*. This is natural. Systems are becoming more software-intensive and software development is moving into more flexible, or Agile, processes. This affects hardware development as well. Furthermore, hardware and systems development face the same challenges as software development. The review showed that the current trends in development, such as an increasing amount of change, complexity of products and shorter product and technology life cycles suggest benefits from Agile development. The suggested approach is incremental co-design involving all engineering disciplines and frequent integration at system level. Based on the review, collaboration between the different engineering disciplines resulted in a number of benefits, such as efficiency, innovativeness leading to significantly different designs compared to existing solutions, and improved system-level optimization.

Testing was frequently mentioned in the primary studies. It was agreed that moving testing forward in the project has benefits, such as avoiding implementing extra features and the cost benefit from identifying mistakes sooner. However, it was found to be challenging to implement early testing activities in hardware development.

Iterative hardware development was found not just to be possible, but to be beneficial to development efficiency. On the other hand, even while the difference is diminishing, the cycle time in hardware development remains longer than in software. This means that during a meaningful cycle of hardware development, the software development goes through multiple iterations. Synchronizing software and hardware development frequently was still identified as advantageous. It helped in focusing on the most important requirements and avoiding over-engineering the technical solution.

During the analysis, a secondary interpretation found an *enforcing cycle* between co-design, testing, and iterative hardware development. Early testing of partial solutions has been seen as an extra cost. Luckily, more intensive, extended and collaborative co-design creates new ideas on how to move testing forward, and how to automate an increasing portion of testing. Automating testing in turn enables even faster iterations in hardware

development. Another roadblock for iterative hardware development has been the long lead times from prototype suppliers. Working with partner-like prototype suppliers, the cycle time can be shortened, and the cadence of development accelerated. In addition, new technology brings other disciplines closer to software development. This in turn spins the wheel forward and creates an enforcing cycle of continuous improvement and innovation.

7.2 Comparison of the case and literature studies

Several similarities were found when the two studies were analyzed together. This strengthens the grounding of the findings. During the analysis, four themes emerged: learning and innovation, whole team approach, emergent process, and remaining challenges. The analysis is summarized in Table 18.

Table 18. Cross-analysis of the two studies.

Theme	**Case Study**	**Systematic Literature Review**	**Summary/Recommendation**
Learning and innovation	Learning about the design was accelerated through up-front prototyping.	Early and extended co-design was experienced to bring many benefits such as more innovative solutions.	Use of fast-paced learning cycles leads to faster and more innovative designs.
Whole team approach	Collective ownership of product and project resulted in improved commitment. Shared goals led to improved communication.	Multidisciplinary nature of development teams resulted in better-optimized solutions. Co-design led to different relationship between developers from multiple disciplines.	In complex environments (such as multidisciplinary development) a single view is not enough to understand the whole system.
Emergent process	The process was improved through Retrospective meetings involving the whole team.	Synthesis found a reinforcing improvement loop between co-design, testing and iterative hardware development.	Scrum provides a framework for process improvement through inspect and adaptation loops. Other Agile methods provide guidance on practices that could be adopted, but they require transformation from the software domain.
Remaining challenges	The need for engineering practices to support multiple prototyping cycles was identified. Conflicts with the rest of the organization lead to large-scale organizational change.	Testing was identified as a key problem to be solved. Development team does not work in isolation, but is dependent on the rest of the organization, and for example, prototype suppliers.	Changing the development method to Agile affects the whole organization, at many different levels. It creates a need for company-wide learning.

Learning and innovation from extended co-design applying frequent prototyping is a shared finding in both studies. Early experimenting with cross-discipline up-front prototyping was the major contributor to fast

learning in the case study. Prototyping revealed gaps in knowledge and accelerated knowledge transfer. The literature study listed several benefits from co-design, such as the degree of innovation. More intensive collaboration with cross-discipline teams clearly accelerates learning. It helps in sharing existing knowledge between disciplines, but also in creating new knowledge. New ways of learning reveal new challenges. The studies identify difficulties in reacting to feedback and cycle cost concerns. These challenges are linked. When the team fully harnesses the learning potential available from the feedback, the cycle cost is justified. In contrast, when the team does not react to feedback, it completely jeopardizes the return on investment from the cycle cost. Concrete example of this include leaving dealing with issues that are found lurking until the product is transferred to mass production. Leaving these issues without attention until the supposed end of the project reduces the predictability of the project dramatically. On the other hand, disciplined fast-paced experimenting has the potential to accelerate learning and innovation dramatically.

Whole team approach, involving the team as a whole in all decision-making was observed to bring several benefits. When people from different engineering disciplines work together toward a shared short-term goal, it improves communication significantly. A major contributor to the improvement is the daily access to other disciplines, enabling the pull information practice mentioned in the case study results. The other discipline can get exactly the information it needs, at exactly the time when it needs it and in the format best suited for the use. Trust between project members was found to be essential for improved communication, but it was challenging to achieve in the case study. Building trust takes time, and in the case study this was only achieved a significant time into the project. Trust is not explicitly mentioned in any of the primary studies in the literature review. However, it is mentioned that short-term goals and shared responsibility create different relationships between disciplines. Furthermore, it is pointed out that blaming and responsibility pushing are avoided with a more collaborative philosophy. This in turn can be interpreted as trust. In addition, the case study identified that the increased whole team involvement resulted in improved commitment. This is not explicitly mentioned in any of the primary studies in the systematic literature study.

The case project began experimenting with Agile development based on the Scrum project management framework. The *emergent process* was continuously improved and refined based on the ideas created in the Retrospective meetings. Improvements happened in several areas, such as communication mechanisms supporting co-design and processes for prototyping. A key finding in the systematic literature review was that these improvements do not happen independently. Co-design, testing, and iterative hardware development together create an enforcing improvement cycle. The case study supported this finding. The case study started with collaborative co-

design. Early testing was improved in cooperation by creating better, more efficient practices for up-front prototyping and integration. All this together shortened the iterations the team felt comfortable with.

While both studies indicated that knowledge from Agile software development methods can be transferred at different levels into other engineering disciplines, they also identified a number of *remaining challenges*. The challenges begin at team level. The rapid cycles require changes in engineering practices. The need for adapted engineering practices was evident based on both studies. Moving testing forward is seen to be effective, but to enable efficient regression testing, the level of automation needs to grow. In the case study, this need was identified, but a solution remained unfound. Some examples of new ways to automate testing are presented in the primary studies of the literature review. One contributor is increased collaboration between software and other disciplines. Programmers can identify opportunities and experiment with automating the tools. The literature review also found much promise in advancing technologies bringing other development closer to software development. For example, in FPGA and ASIC development, software testing techniques become more readily available. Challenges were equally evident outside the team. When one function changes the way of working, it conflicts with many of the organization's existing processes. An example identified in both studies is the relationship with prototype suppliers. Established partner-like relationships between the suppliers made it possible to have 24h delivery of physical prototypes in the case study. The official bureaucratic process for prototyping would have taken weeks for each prototype round. Both studies also identified challenges beyond the immediate stakeholders. The next sub-chapter looks at the implications of Agile development for the whole organization.

7.3 Further discussion on organizational implications

The development process cannot be changed without affecting the rest of the organization. We will first take a look at the development organization. A common development organization structure is divided into lines based on engineering discipline. An Agile system development team relies on the opposite structure. The team has members from different engineering disciplines. A conceptual framework presenting how the organization moves from line organization toward Agile System Development teams is illustrated in Figure 18. The framework presents four stages: Waterfall, hybrid, single-discipline Agile and multi-discipline Agile. The stages are used in the industry according to the author's observation, and their order describes how well they support the idea of a self-organizing team. It is assumed that the more cross-disciplined the team is, the more self-organizing it can be. The framework characterizes the different stages by how they differ in three aspects: team

structure, method of co-working, and motivation for prototyping. The three aspects were chosen because they were the most visible changes based on both studies in this thesis. The framework defines a Waterfall process as an example of the "traditional" project management of system product development. In this model, the communication is mainly done using written documentation at pre-defined stages. Development in multiple engineering disciplines follows a plan and results from different lines of development are integrated at the end, or at a few intermediate points. The hybrid process model combining Waterfall and Agile processes can be seen as the initial step toward Agility. At this stage, one or more disciplines follow a Waterfall process, or the Waterfall process is seen as the grand process above the Agile development. This stage still has the rigidity of the bureaucratic control-oriented culture. Organizations that have moved into the third stage, single-discipline Agile, have organized development into Agile teams. However, these are still mainly formed according to engineering discipline. The communication is directed via the role of a system architect or similar, but the synchronization of different disciplines is done much more frequently. The last stage in the framework involves organizing the whole development around cross-discipline system development teams. The synchronization of different engineering disciplines is done in real-time, as the team commits to solving problems together. This is possible because they have skills and knowledge from all disciplines. The framework calls this category multi-discipline Agile. The case study is positioned between single-discipline Agile and multi-discipline Agile. The teams adopted the practice of the communication bridge in order to have members from different disciplines. This clearly showed the need for intensive collaboration between the disciplines. Team members also confirmed that they would like to be full-time members of a system development team.

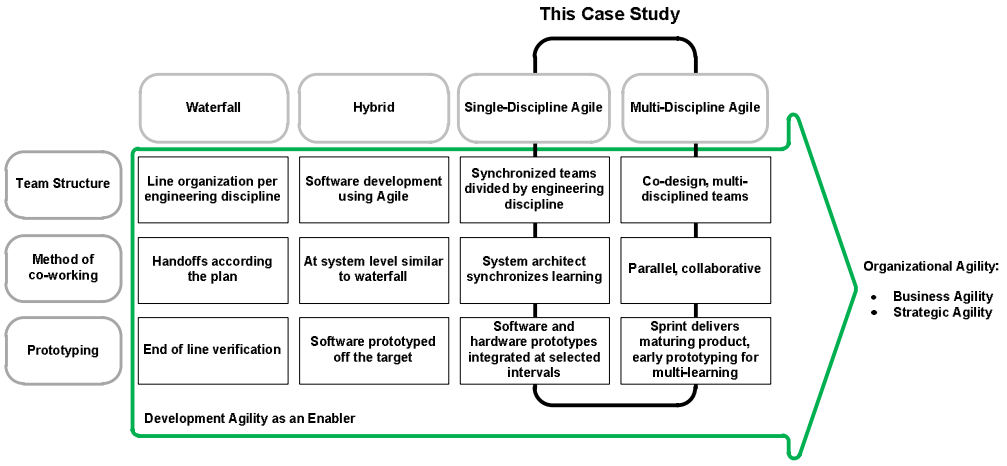

Figure 18. A conceptual framework for development Agility.

However, changing the structure of development organization does not solve everything. Development Agility cannot be the goal itself and development Agility alone is not enough, either. As illustrated in Figure 18, the organization

should move forward from development Agility. Kettunen and Laanti (2008) propose a model for thinking about organizational Agility from the perspective of the company's goals. The model is further explained in Laanti (2012). It describes how an Agile organization works. *Goals* define what the organization really wants to achieve. *Means* are the practices the company has chosen to implement in order to achieve those goals. *Enablers* are the factors, conditions and abilities that make achieving the goals possible through the selected means. The enablers, means and goals model is illustrated in Figure 19. To see how the results from this research fit the model, we choose the goal to be ambitious: "*sustainable competitiveness in a turbulent environment.*" This goal is derived from the challenges presented in chapter 2.3:

- Products need to get to market faster
- Increasing amount of change (or learning) during the development
- Products to be developed are getting more complex

Examples of means identified in the studies are incremental and iterative co-design and a flexible, modular system-level platform. In the context of this thesis, it is important to notice that the means and practices, are shared by multiple engineering disciplines. They describe Agile co-design at the system level. The studies found a number of enablers as well, such as technology improvements in many areas. Development Agility is only a means to an end. By combining the goals, means, enablers model and findings of this thesis, we can clearly notice that organization-wide strategic alignment is needed, as can be seen in Figure 19.

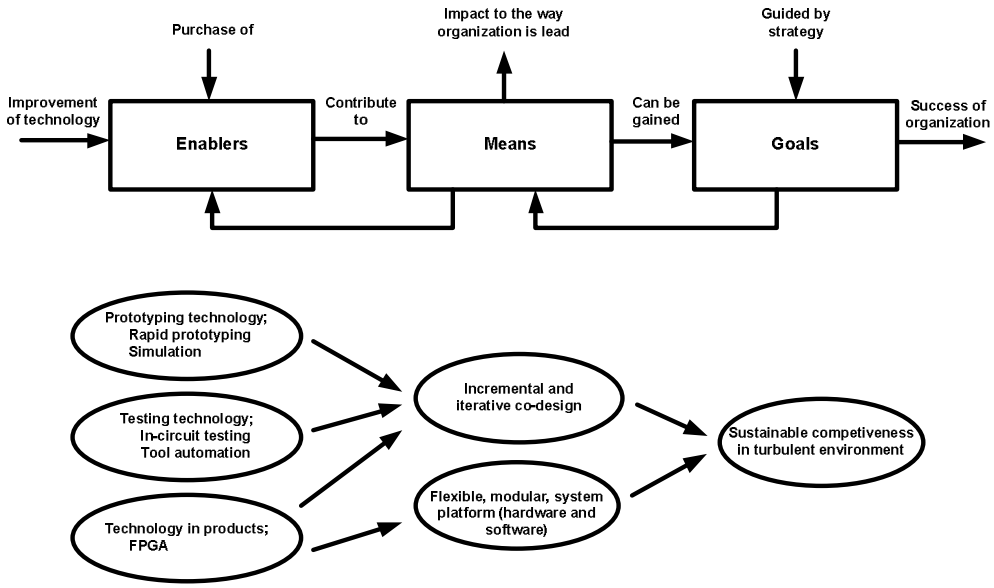

Figure 19. Iterative and incremental co-design modeled in organizational improvement context (Kettunen and Laanti, 2008; Laanti, 2012).

7.4 Answering the research questions

The overarching research problem was:

> *Can the knowledge from Agile software development be transferred to supplement flexible product development processes and accelerate their adoption in other engineering disciplines and system development?*

The research problem was further divided into three research questions. They are answered in the next sub-sections.

7.4.1 Research question 1

> *Can knowledge from Agile software development be transferred to non-software new product development?*

The first research question takes a look at the research problem on a very generic level. In the case organization, knowledge from Agile software development and the Scrum framework was adopted successfully in a non-software development project. Based on this research, the knowledge is directly transferable on a higher level. The Agile Manifesto and Scrum framework can be adapted with very few modifications. The work can be done mainly at the level of definitions. For example, the concepts of customer role and potentially shippable product need to be adapted for domains outside software development. More specific engineering practices, labeled as Agile engineering practices and mostly coming from Extreme Programming, need more thorough transformation. Nevertheless, the need to learn from practices such as continuous integration, test-driven development, collective ownership and pair programming (working) was evident during this research.

The systematic literature review showed that the documented knowledge on the topic is limited. However, studies were found showing that knowledge transfer has been successfully implemented in other organizations as well, supporting the result from the case study. As a conclusion, we can say that knowledge from Agile software development can be transferred to other development domains. It is valuable to remember that the knowledge transfer from Agile software development can happen at many levels.

7.4.2 Research question 2

What are the implications of introducing concepts familiar from Agile software development in the development of physical products?

The second research question looks at the possible positive and negative implications that the introduction of Agile development brings. Both studies found Agile development to bring several benefits, such as accelerated learning, improved commitment, efficiency, and more innovative solutions. Mainly, this contributed to extended, iterative co-design. In the case project, the short time-boxed experiments with physical prototypes revealed gaps in knowledge. This discovery guided the teams to acquire the needed knowledge immediately. The involvement of the whole team in product and process design led to increased commitment and efficiency. The growing degree of innovativeness can be argued to be the result of having people from multiple engineering disciplines closely collaborating on a daily basis. This reduces misconceptions but also helps in multi-learning, learning to think from the perspective of other disciplines and learning new ways of learning together.

On the other hand, Agile development causes a need for deeper change in engineering. Iterative development makes the need for continuous testing clearly visible. When thorough learning is needed frequently, the level of test automation needs to grow. Testing in iterative hardware development was seen as a key challenge, but concrete examples of solutions were lacking. For software development, this is fairly well understood, and tools and technologies are available, but for other engineering disciplines, the solution is not so straightforward. Collaboration between engineering disciplines, on the other hand, was seen as creating solutions for more efficient testing and this changed the attitude toward fast iterations in a more positive direction.

7.4.3 Research question 3

Does Agile development impose larger organizational implications?

The final research question shifts the focus to the surroundings of the development function. Development does not happen in isolation. The change in the way of working affects the immediate stakeholders and eventually the whole organization. Close proximity effects were identified in the case study. For example, the case project experienced difficulties because the product management was not officially included in the process change. Furthermore, both studies identified the need for a change in co-working with external

parties such as other development sites and prototype suppliers. This topic also came up in the literature on flexible product development. The early involvement of senior management in the development process, a change in managerial style and a company-wide focus on learning were mentioned as examples of these changes.

Agile development can only be seen as a means to achieve a company's fundamental goal. The fundamental goal is to be successful, both in the short and long term. Agile development alone cannot be the whole answer. For a better outcome, company-wide alignment is needed.

7.5 Limitations

The study has several limitations that need consideration. The case study was a single isolated case. In this light, it is fair to say that there are limitations in generalizability. It is worth mentioning that every environment, project, and project organization is different. In addition, the case study and primary studies of literature review presented results from the introductory phase of the Agile process. Therefore, any long-term implications remain to be studied.

The case project was started without a pre-study of existing knowledge or a thorough research plan. This was justified because an opportunity for observation in the industrial setting was available. Further, the results turned out to be trustworthy and to support existing data. It must be granted that with more time for pre-research, there might have been chances to identify important matters, such as investment in testing practices.

Because of the distance and worst possible time difference, it was impossible to observe how the teams worked on a daily basis (e.g. self-organizing team or Daily Scrum). Providing continuous coaching for teams was also impossible. Thesis writing and post-project interviews happened over a year after the project. All of the people involved in the case project had continued to practice Agile development and project management practices. For this reason, it was sometimes difficult to remember what was done during the case project, and what was only done later. However, the key findings presented were consistent and distributed throughout data from different sources (documents, presentations, developer and technical lead interviews) and thus data supporting these findings can be said to be saturated. This leads to the conclusion that presented findings are grounded. They also support the earlier research.

There is a considerable risk of bias when the researcher works in the company in which the study takes place. This bias was reduced by the fact that the author was not an active member of the project. In addition, several versions of the thesis were reviewed by the participants.

The existing literature on applying Agile methods to hardware development is scarce. This forced the selection of primary studies from quite a broad area. This together with the questionable academic quality of the studies resulted in rather scattered data. The existing literature mainly consists of lessons learned papers based on expert opinion. Furthermore, the search for studies was undertaken using keywords more familiar to the software industry. Therefore, it is likely that it missed studies that describe fast-paced incremental and iterative hardware development, but do not reference Agile software development. On the other hand, this research focuses on knowledge transfer from Agile software development and therefore the selection of keywords is justified. It is also worth noticing that none of the studies reported failure. It is fair to expect that failures do exist when companies have started investing in up-front prototyping.

8 Conclusions

This chapter takes a final look at the entire research project. It presents a brief summary of the thesis and proposes future research topics as identified during the research.

8.1 Summary and conclusions

The objective of this research was to find out if knowledge transfer from Agile software development to the hardware development domain is possible, and the implications of such an effort. A two-part study was conducted: a case study in an industrial setting and a systematic literature review. The results from both parts supported each other, and also aligned with the existing literature.

The data from the two studies forms a solid answer to the overarching research problem. Knowledge from Agile software development can, and should, be taken into account for other disciplines as well. Although knowledge on emergent processes for product development in general has been available for decades, it is Agile software development that has been rapidly adopted in the industry with encouraging results. There is promise in the knowledge transfer; especially due to the increased software intensity in products that companies develop. Starting iterative hardware development after only brief Agile development training was straightforward in the case organization. Therefore, we can conclude that knowledge from Agile software development can accelerate the adoption of flexible process models. The process itself brought several benefits, such as accelerated learning and innovation, a more collaborative whole team approach, and continuous improvement of the emergent process. However, Agile development requires some changes in traditional hardware development. The most obvious change is the process for using prototyping during the development. In Agile development, the use of prototypes is much more intense, and requires a streamlined process and a partner-like relationship with prototype suppliers. Another example is testing activity. Testing becomes an integrated activity throughout the development, and new testing practices are needed.

The primary contribution of this study is adding knowledge from the field, by practitioners, on knowledge transfer from Agile software development to the hardware development domain. Research has been conducted on combining Waterfall-like processes for hardware development with Agile software development (Karlström and Runeson, 2006). In contrast, this thesis presented how hardware development can benefit from following Agile development methods. These findings should encourage others to experiment with Agile methods in different environments, not limited to software

development. Agile embedded software development is getting attention already, and the natural continuum is to study Agile co-design in system development. The systematic literature review was only able to find only a few studies on the subject. This further emphasizes the importance of the contribution. All in all, this study gives a partial answer to Smith's (2007) request for evidence from the field regarding the use of Agile development methods outside the software development realm.

The practical contribution of this work can be found beneficial for all the roles in an organization. It encourages organizations to leverage co-learning by continuously transferring knowledge across the engineering discipline boundaries. It provides guidance on how to get started in implementing iterative system-level development using knowledge from Agile software development. On a higher level, a framework for development organization Agility was proposed according to the level of co-design Agility and collaboration. Secondly, it was argued that development Agility is not the end objective, but rather a means to achieve organizational goals. Therefore, this thesis also contributes to the understanding of development Agility being a source of organization-wide change.

8.2 Future research

Both the systematic literature review and the case study clearly show that we still need more explorative studies in this area. In order to grow the knowledge on the use of Agile methods in systems development, several more focused areas of interest for future research were identified.

When the whole development department moves into a fast iterative rhythm, it is inevitable that it affects the whole organization. This leads to interest in Agile enterprise process integration. Based on the experience from the case study, first steps could include the early involvement of people responsible for industrialization. Often the industrialization phase is the responsibility of an independent organization, and the transfer of responsibility from product development to industrialization is seen as a single event, and can be quite bureaucratic. The model for involving larger project organizations needs exploring.

A particular area of interest is the economics of up-front prototyping strategies. Up-front prototyping in system development is at the heart of Agile development. Can you always justify up-front prototyping? Can we put a price tag on time? How do we calculate the saved time, when we do not have data from the traditional approach? The process of balancing the cycle cost of a given product and the optimal prototyping frequency needs guidelines. For example, can you quantify the amount of missing knowledge versus the cost of acquiring it? In advanced development, it would be interesting to find ways to

justify prototyping parallel solutions and making a late commitment to one optimal one. A cost analysis of the number and maturity of prototypes would be interesting to study. How does innovation fit into the picture? And finally, what about situations where we do not know what we do not know?

Moving testing forward in system development is a major challenge on its own. Both studies found this to be one of the most important areas for improvement. Today Agile testing practices (such as test automation and test-driven development) are thoroughly documented for software development, including embedded software (Grenning, 2014). While it is fairly straightforward to adapt Agile project management practices to hardware development, very little is written about specific practices suitable, for example, for test-driven hardware development. Here again, a framework for finding the balance between investment in test automation and saved time and quality improvement is an interesting topic. It is easier to justify test automation in software development, as it is very probable that software changes after first being released to production. Today, despite the continuously shortening life cycles, hardware is still less likely to change frequently while in production.

It is a fairly common misconception that there is no role for written documentation in Agile development. This is not true, and the minimal amount of written documentation was found to be problematic in the case study. The study took place in an organization that officially followed a sequential Stage-Gate process and had a CMMI initiative. In the case project, these obligations were overlooked. Research should consider the appropriate amount and method of documenting for future development. Research exists on the integration of Agile development and legitimacy processes and quality and improvement models, for example, Stage-Gate (Karlström and Runeson, 2006), ISO 9001 (Cockburn, 2005, Namioka and Bran, 2004 and Stålhane and Hanssen, 2008) and CMMI (Sutherland, Jakobsen and Johnsson, 2007). This work should be continued from the practical perspective, concentrating on a documentation approach that helps the organization and at the same time satisfies the certified quality assurance process.

At the time of writing this thesis, the topic of Agile methods for hardware development domain is getting more attention. At a recent embedded systems conference, there were three talks presenting benefits of Agile development in hardware and systems development (Punkka, 2012; Schoenderwoort, 2012; Liberty, 2012). Agile, iterative, ASIC development has also gotten attention in the past couple of years (Johnson, 2012; Leisgang, 2012). Studies have also been conducted on the applicability of Scrum even in non-technical fields such as venture group, church, and management teams (Sutherland, Sutherland and Hegarty 2009; Sutherland and Altman, 2009; Barton, 2009; Figueiredo, 2009). The results have been encouraging, and the work should be continued.

Knowledge transfers should not be considered as one-time, one-direction events. Rather, the exchange of knowledge between disciplines and domains should be continuous.

9 References

(Barton, 2009) Barton, Brent, All-Out Organizational Scrum as in Innovation Value Chain, Proceedings of the 42nd Hawaii International Conference on Systems Sciences (HICSS), 2009.

(Larman and Basili, 2003) Larman, Craig and Basili, Victor R., Iterative and Incremental Development: A Brief History, IEEE Computer, June, 2-11, 2003.

(Beck, 2000) Beck, Kent, Extreme Programming Explained, Addison-Wesley, 2000.

(Beck and Andres 2004) Beck, Kent and Andres, Cynthia, Extreme Programming Explained: Embrace Change, 2nd Edition, Addison-Wesley Professional, 2004.

(Boehm, 2006) Boehm, Barry, Some Future Trends and Implications for Systems and Software Engineering Processes, Systems Engineering, Vol.9, No. 1, 1-19, Wiley Periodicals, Inc, 2006.

(Brereton et.al., 2007) Brereton, Pearl, Kitchenham, Barbara, Budgen, David, Tuner, Mark and Khalil, Mohammed, Lessons from Applying the Systematic Literature Review Process within Software Engineering Domain, The Journal of Systems and Software, No.80, 571-583, 2007.

(Clay and Smith, 2000) G. Thomas Clay and Preston G. Smith, Rapid Prototyping Accelerates the Design Process, Machine Design Magazine (available at http://www.machinedesign.com/), March 9, 166-171, 2000.

(Cockburn and Highsmith, 2001) Cockburn, Alistair and Highsmith, Jim, Agile Software Development: The Business of Innovation, IEEE Computer, September, 120-127, 2001.

(Cockburn, 2001) Cockburn, Alistair, Agile Software Development, Addison Wesley, 2001.
(Cockburn, 2005) Cockburn, Alistair, Crystal Clear – A Human-Powered Methodology for Small Teams, Addison Wesley, 2005.

(Coghlan and Brannick, 2001) Coghlan, David and Brannick, Teresa, Doing Action Research in Your Own Organization, Sage Publications, Inc., 2001.

(Cohn, 2006) Cohn, Mike, Agile Estimating and Planning, Prentice Hall, 2006.

(Cohn, 2010) Cohn, Mike, Succeeding with Agile, Addison Wesley, 2010.

(Cooper, 1986) Cooper, Robert G., Winning at New Products, 1st Edition, Addison Wesley, 1986.

(Cooper, 2001) Cooper, Robert G., Winning at New Products, 3rd Edition, Perseus Publishing, Cambridge, Mass., 2001.

(Cooper, 2009) Cooper, Robert G., How Companies are Reinventing Their Idea-to-Launch Methodologies, Research-Technology Management, Vol.52, No.2, March-April, 47-57, 2009.

(Cooper and Edgett, 2009) Cooper, Robert G. and Edgett, Scott J., Lean, Rapid and Profitable New Product Development, Booksurge Publishing, 2009.

(Cunningham, 1997) Cunningham, J. Barton, Case Study Principles for Different Types of Cases, Quality & Quantity No.31, 401-423, Kluwer Academic Publishers, Netherlands, 1997.

(Doz and Kosonen, 2008) Doz, Yves and Kosonen, Mikko, Fast Strategy: How Strategic Agility Will Help You Stay Ahead of the Game, Pearson Education Limited, 2008.

(Figueiredo, 2009) Figueiredo, Alexandre Magno, An Executive Scrum Team, Agile Conference 2009.

(Fowler, 2006) Fowler, Martin, Using an Agile Software Process with Offshore Development, (online, available at http://martinfowler.com/articles/agileOffshore.html), 2006.

(Grenning, 2011) Grenning, James W., Test-Driven Development for Embedded C, The Pragmatic Bookshelf, 2011.

(Hamel and Prahalad, 1994) Hamel, Gary and Prahalad, C.K., Competing for the Future, Harvard Business School Press, Boston, Massachusetts, 1994.

(Highsmith, 1999) Highsmith, James, A., Adaptive Software Development, Dorset House Publishing., Inc., 1999.

(Highsmith, 2001) Highsmith, Jim, Manifesto for Agile Software Development, (online, available at http://agilemanifesto.org/), 2001.

(Highsmith, 2002) Highsmith, Jim, Product Development and Agile Methods, The Cutter Edge (online, available at: http://www.cutter.com/research/2002/edge020528.html), 2002

(Highsmith, 2007) Highsmith, Jim, Panel discussion at Agile Conference 2007, Washington D.C., Agile Conference 2007.

(Highsmith, 2009) Highsmith, Jim, Agile Project Management: Creating Innovative Products, 2nd Edition, Addison-Wesley Professional, 2009.

(Hutchings et al., 1993) Hutchings, Tony, Hyde, G. Michael, Marca, David, and Cohen, Lou, Process Improvement that Lasts: Integrated Training and Consulting Methods, Communications of ACM, Vol.36, No.10, 105-113, October 1993

(Johnson, 2012) Johnson, Neil, TDD and a New Paradigm for Hardware Verification, Agile Conference, 2012. Presentation available at:

http://www.agilealliance.org/files/session_pdfs/TDD%20and%20a%20new%20paradigm%20for%20hardware%20verification.pdf

(Karlström and Runeson, 2006) Karlström, Daniel and Runeson, Per, Integrating Agile Software Development into Stage-Gate Managed Product Development, Empirical Software Engineering, Vol.11, No.2, 203-225, June, 2006.

(Katzenbach and Smith, 2003) Katzenbach, Jon and Smith, Douglas, The Wisdom of Teams: Creating the High-Performance Organization, Harper Business, 2003.

(Kettunen and Laanti, 2008) Kettunen, Petri and Laanti, Maarit, Combining Agile Software Projects and Large-scale Organizational Agility, Software Process Improvement and Practice, No.13 183-193, 2008.

(Kitchenham, 2007) Kitchenham, Barbara, Guidelines for performing Systematic Literature Reviews in Software Engineering, 2007.

(Laanti, 2012) Laanti, Maarit, Agile Methods in Large-Scale Software Development Organizations – Applicability and Model for Adoption, Doctoral Thesis, Department of Information Processing Science, University of Oulu, 2012.

(Leisgang, 2012) Leisgang, Tobias, How to Play Basketball with a Soccer Team? Making IC development more agile, Agile Conference, 2012. Presentation available at: http://www.agilealliance.org/files/session_pdfs/Agile2012_SoccerWithABasketballTeam.pdf

(Liberty, 2012) Liberty, Matt, Agile Hardware, Embedded Systems Conference, Boston, 2012.

(MacCormack, Verganti and Iansiti, 2001) MacCormack, Alan, Verganti, Roberto and Iansiti, Marco, Developing Products on "Internet Time": The Anatomy of a Flexible Development Process, Management Science, Vol.47, No.1, 133-150, January, 2001.

(Miles and Huberman, 1994) Miles, Matthew and Huberman, Michael, Qualitative Data Analysis, 2nd Edition, SAGE Publications, 1994.

(Manning and Rising, 2005) Manns, Mary Lynn and Rising, Linda, Fearless Change: Patterns for Introducing New Ideas, Pearson Education, Inc., 2005.

(Mishra and Mishra, 2009) Mishra, Deepti and Mishra, Alok, Effective Communication, Collaboration, and Coordination in eXtreme Programming: Human-Centric Perspective in a Small Organization, Human Factors and Ergonomics in Manufacturing, Vol.19, Issue 5, 438-456, 2009.

(Namioka and Bran, 2004) Namioka, Aki and Bran, Cary, eXtreme ISO ?!?, Companion to the 19th annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), 2004.

(Nerur, Mahapatra and Mangalaraj, 2005) Nerur, Sridhar, Mahapatra, RadhaKanta and Mangalaraj, George, Challenges of Migrating to Agile Methodologies, Communications of the ACM, May/Vol.48, No.5, 72-78, 2005.

(Noblit and Hare, 1988) Noblit, G.W. and Hare, R.D. Meta-Ethnography: Synthesizing Qualitative Studies. Sage Publications, 1988.

(Ogunnaike and Ray,1994) Ogunnaike, A. Babatunde and Ray, W. Harmon, Process Dynamics, Modeling and Control, Oxford University Press, Inc., 1994.

(Ovesen, 2012) Ovesen, Nis, The Challenges of Becoming Agile – Implementing and Conducting Scrum in Integrated Product Development, PhD Thesis, Aalborg University, 2012.

(Paasivaara, 2005) Paasivaara, Maria, Communication Practices in Inter-Organisational Product Development, Doctoral Dissertation, Helsinki University of Technology, Department of Computer Science and Engineering, 2005.

(Patton, 2002) Patton, Michael Quinn, Qualitative Research & Evaluation Methods, 3rd Edition, Sage Publications, California, 2002.

(PMI, 2010) 2010 PMI's Pulse of profession, (online, available at: http://www.pmi.org/~/media/PDF/Home/Pulse%20of%20the%20Profession%20White%20Paper%20-%20FINAL.ashx), 2010.

(Punkka, 2012) Punkka, Timo, Agile Hardware and Co-design, Embedded Systems Conference, Boston, 2012.

(Potts. 1993) Potts, Colin, Software-Engineering Research Revisited, IEEE Software, September 1993, 19-28.

(QSMA, 2008) Agile Impact Report, QSM Associates, 2008. (Online, available for order at http://www.qsma.com/agile-impact-report.html)

(Ratner and Harvey, 2011) Ratner, Ian and Harvey, Jack, Vertical Slicing: Smaller is Better, AGILE'11 Proceedings of the Agile Conference, IEEE Computer Society, Washington DC, 2011.

(Reinertsen, 1997) Reinertsen, Donald G., Managing the Design Factory, A Product Developers Tool Kit, Simon & Schuster Ltd, 1997.

(Reinertsen, 2009) Reinertsen, Donald G., The Principles of Product Development Flow – Second Generation Lean Product Development, Celeritas Publishing, Redondo Beach, California, 2009.

(Royce, 1970) Royce, Winston, Managing the Development of Large Software Systems, Proceedings of IEEE WESCON No.26, August 1970.

(Schwaber and Beedle, 2002) Schwaber, Ken, and Beedle, Mike, Agile Software Development with Scrum, Prentice Hall 2002.

(Schwaber, 2004) Schwaber, Kent, Agile Project Management with Scrum, Microsoft Press, 2004.

(Schooenderwoert, 2012) Van Schooenderwoert, Nancy, Agile Hardware Development: Why Lean and Agile Principles Work Here, Embedded Systems Conference, Boston, 2012.

(Smith, 1990) Smith, Preston, G., Fast-Cycle Product Development, Engineering Management Journal, Vol.2, No.2, June, 328-338, 1990.

(Smith and Reinertsen, 1997) Smith, Preston, G., and Reinertsen, Donald G., Developing Products in Half the Time; New Rules, New Tools, 2nd Edition, John Wiley & Sons Inc, 1997.

(Smith, 2007) Smith, Preston, G., Flexible Product Development, Jossey-Bass, 2007.

(Sosa et. al., 2002) Sosa, Manuel, Eppinger Steven, Pich, Michael, McKendrick, David, and Stout, Suzanne, Factors That Influence Technical Communication in Distributed Product Development: An Empirical Study in the Telecommunications Industry, IEEE Transactions on Engineering Management, Vol.49, No.1, February, 45-58, 2002.

(Stringer, 1999) Stringer, Ernest T., Action Research, 2nd Edition, Sage Publications, Inc., 1999.

(Stålhane and Hanssen, 2008) Stålhane, Tor and Hanssen, Geir Kjetil, The Application of ISO 9001 to Agile Software Development, Proceedings of the 9th International Conference on Product-Focused Software Process Improvement (PROFES), 2008.

(Sutherland, Jakobsen and Johnsson, 2007) Sutherland, Jeff, Jakobsen, Carsten Ruseng and Johnsson, Kent, Scrum and CMMI Level 5: The Magic Potion for Code Warriors, Agile Conference, 2007.

(Sutherland, Sutherland and Hegarty 2009) Sutherland, Arline, Sutherland, Jeff, Hegarty, Christine, Scrum in Church: Saving the World One Team at a Time, Agile 2009 Conference, 2009.

(Sutherland et.al., 2007) Sutherland, Jeff, Viktorov, Anton, Blount, Jack, Puntikov, Nikolai, Distributed Scrum: Agile Project Management with Outsourced Development Teams, Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS), 2007.

(Sutherland and Altman, 2009) Sutherland, Jeff and Altman, Igor, Take No Prisoners: How a Venture Capital Group Does Scrum, Agile 2009 Conference, 2009.

(Sutherland, Viktorov and Blount, 2006) Sutherland, Jeff, Viktorov, Anton and Blount, Jack, Adaptive Engineering of Large Software Projects with Distributed/outsourced Teams, Proceedings of the Sixth International Conference on Complex Systems, New England Complex Systems Institute, 2006.

(Tabaka, 2008) Tabaka, Jean, Collaboration Explained, Facilitation Skills for Software Project Leaders, Addison-Wesley, 2008.

(Takeuchi ja Nonaka, 1986) Takeuchi, H. and Nonaka I., The New Product Development Game, Harvard Business Review, Vol. 64, No. 1, 137-146, 1986.

(Thomke and Reinertson, 1998), Thomke, Stefan and Reinertson, Donald, Agile Product Development: Managing Development Flexibility in Uncertain Environments, California Management Review, Vol.41, No.1, Fall, 8-30, 1998.

(Versionone, 2013) VersionOne, 7th Annual State of Agile Development Survey, 2013. (Online, available at: http://www.versionone.com/pdf/7th-Annual-State-of-Agile-Development-Survey.pdf

(West and Grant, 2010) West, Dave and Grant, Tom, Agile Development: Mainstream Adoption Has Changed Agility, Forrester Research Inc., January, 2010.

(Yin, 1994) Yin, Robert K., Case Study Research; Design and Methods, 2nd Edition, Sage Publications Inc., California, 1994.

APPENDIX A – Data extraction from primary studies

Summary of themes in selected studies (A)				
ID	147	125	107	101
Author(s)	Doran, Hans Dermot	Van Schooenderwoert, Nancy and Morsicato, Ron	Suhaib, Syed, Mathaikutty, Deepak and Shukla, Sandeep	Boehm, Barry
Title	XP: Good for Anything Other than Software Development	Taming the Embedded Tiger – Agile Test Techniques for Embedded Software	Extreme Formal Modeling (XFM) for Hardware Models	Some Future Trends and Implications for Systems and Software Engineering Processes
Type of study	Experience report	Experience report	Experience report	Technical paper
Year of pub	2003	2004	2004	2006
Agile method	XP	XP	XP	Generic
Engineering discipline	Co-design	Hardware Co-Design Embedded SW	Hardware	Co-design
Co-design	"…an unusually high number of interacting features and elements could be completed in a short period of time." (pg.3)	"Our [SW team's] relationship with the hardware group was qualitatively different from what we've observed in non-Agile projects. [isolating defects and co-working]" (pg.6)		"…organizations that have used LCO and LCA milestones and their Feasibility Rationale pass/fail criteria to synchronize hardware and software architecture definition have found this highly advantageous." (pg.14) "Systems engineers without software experience would minimize computer speed and storage costs and capacities, which causes software costs to escalate rapidly." [increasing integration of software end systems engineering] (pg.3) "Negotiation of priorities for requirements involves not only participation from users and acquirers on each requirement's relative mission or business value, but also participation from systems and software engineers on each requirement's relative cost and time to develop and difficulty of implementation." (pg.4)

				"…slack must be incorporated into the process to keep the ensemble of feature teams synchronized and stabilized." (pg.15)
Testing	"…in the area of system design, it did not appear to occur to anybody that one could determine how a system is to be tested before it is actually built." (pg.2)	"We would have been overwhelmed without [dual targeting] the ability to quickly isolate hardware problems." (pg.2) "…hardware unit tests call production routines in the module that directly access hardware... These hardware unit tests are very valuable for groups outside the software team – electrical engineers, test personnel, and production technicians." (pg.4-5)	"Whenever a property fails to validate, it usually is straightforward to find the bug as it must be related to the latest additions" (pg.3) "To refactor problems, to update tests after a bug is found, and to work in pairs are also principles that are as beneficial to the capturing of formal methods as they are for common programming projects." (pg.2) "…this rule [test-first] maps to specifying the linear time property before writing the abstract model (property-driven approach)." (pg.2)	
Iterative HW	"…despite the successful completion of initial iterations, the generation of a production-ready design, which consisted of the correction of two minor issues, suffered from severe problems. …design hadn't been checked out of the repository properly…" (pg.2) "…suppliers didn't perform as promised, which led to delays of up to over a month in getting the prototypes ready." (pg.2)	"In embedded development, the hardware is always changing. It evolves in steps that the software must support. This dovetails nicely with iterative software development." (pg.2)	"At the start of each iteration the goals are identified and written down in the form of 'user stories' – individual cards that point out specific implementation details and requirements."(pg.2) Example: "'Each instruction executes in a certain order'" (pg.3) "In the conventional approach, however, the abstract model tends to contain much more functionality than specified, but less properties than needed as there is no mechanism that provides for the exposure of all properties contained in the specification." (pg.6)	"This often means that several software increments may be fielded between major hardware increments." (pg.14) "It [the spiral model| is a risk-driven set of concurrent prototyping, analysis, and stakeholder renegotiation activities leading to a best-possible redefinition of the plans and specification to be used by the stabilized development team for the next increment." (pg. 14) "…continue to reduce but not eliminate the differences between hardware and software processes". (pg.14)

	“Getting a board from finished design to actual prototype in ten days, a long period in XP terms, costs a significant amount of money and demands that all suppliers in the chain optimise their efforts.” (pg.2)		“This wholesome approach often has the problem that (i) the inconsistency in the properties or mistakes in capturing the intended property are found late, (ii) the synthesis of automata may explode in size when everything is considered together. .. It might be better and more feasible to construct the model by hand incrementally…” (pg.1)	
Ungrouped quotes	“It appears that developers of a certain, median, experience level are required to apply XP successfully.” (pg.3)			
Observations	No data, not even qualitative quotes.	Briefly describes incremental development of hardware, stating that hardware is always changing.	States that earlier behavior can be checked, but does not explain how this is done. Cases to test the method, such as the ISA bus and DLX pipeline.	Future trends guide to concurrent development and more Agile approach.

Summary of themes in selected studies (B)				
ID	97	84	59	54
Author(s)	Chae, Heeseo, Lee, Dong-hyun, Park, Jiyong and Peter, Hoh	Cordeiro, Lucas, Barreto, Raimundo, Barcelos, Rafael, Oliveira, Meuse, Lucena, Vincente and Maciel, Paulo	Kettunen, Petri and Laanti, Maarit	Paelke, Volker and Nebe, Karsten
Title	The Partitioning Methodology in Hardware/Software Co-design Using Extreme Programming: Evaluation through the Lego Robot Project	Agile Development for Embedded Systems: A Platform-Based Design Approach	Combining Agile Software Projects and Large-scale Organizational Agility	Integrating Agile Methods for Mixed Reality Design Space Exploration
Type of study	Experience report	Technical paper	Technical paper	Experience report
Year of pub.	2006	2007	2007	2008
Agile	XP	Scrum, XP	Generic	Scrum
Discipline	Co-design	Co-design	Co-design (may be more later if scope onion is enlarged)	Hardware, co-design
Co-design	"Through adding a guide line of extreme programming to the advantage of co-design, the synergy effect of general process is expected." (pg.1) “However, in fact, there are frequent cases in many R&D laboratories that choosing the processor, designing the hardware and transferring it to software group without communication or interaction” (pg.1) “The important part of the life cycle is iterating and implementing phase which is previous phase of firm HW/SW decision” (pg.2) (the HW/SW partitioning phase is to include iterations)	“Based on this context [embedded characteristics], we propose a development methodology based on the Agile principles such as adaptive planning, flexibility, iterative and incremental approach…” (pg.1) “Slight changes were needed to: (i) adapt processes and tools to optimize the product’s design rather than take paths that lead to designs that have no chance of satisfying the constraints, (ii) support software and hardware development through a comprehensive flow from specification to implementation, (iii) instantiate the system platform based on the application constraints rather than over-design a platform instance for a given product, and	“…it is not enough to focus on the team and project-level dimensions alone, as would a typical application of Agile software methodologies.” (pg.3.)	“Since both types of experts and end-users are involved in the following activities their insight and expertise is available when required and no extensive documentation is required.” (pg.4) “…the multidisciplinary nature of the development teams allows to consider a wide area of expertise and knowledge without large overheads. …[different] aspects can be considered in the design process without the need for extensive documentation” (pg.6)

		(iv) use system platform to conduct various design space exploration analyses for performance." (pg.8)		
Testing	"…the more hardware/software error detection in testing process is delayed, the more time to correct…" (pg.3) "Development cost and error rate is reduced by communicating about these changing issues between hardware and software development groups. " (pg.5) "HW verification is needed to complement the weak point XP cannot cover…" (pg.6) "…development costs of a project can be reduced…" (pg.3)			
Iterative HW			"A more Agile way of doing this product development is:" A) "Common systems engineering to create a modular, flexible product architecture" B) "The need for different hw plug-in unit variants is proactively anticipated (Flexible product architecture)" C) "The software architecture is designed for easy incorporation of new hardware unit types, based on standardized hardware/software interfaces" D) "the related software and hardware projects are managed in close cooperation" E) "integration proceeds smoothly in an iterative fashion" (pg. 6)	"The vague/ambiguous definition of a solution at the beginning suggests an iterative and evolutionary development approach." (pg.3) "We propose to conduct initial design iterations on the interface and its behaviour and components using even more lightweight artefacts in the initial iterations, e.g. design sketches, paper prototypes and mock-ups." (pg.5)

				"Once the design becomes sufficiently stable to identify required hardware and concrete interaction techniques the design representation can be switched to (partial) implementations, resulting in the conventional scrum process." (pg.5) "…this exploratory phase should be extended and used to establish user requirements and to generate alternative MR designs…" (pg.4) "A preliminary analysis with regards to technical feasibility, development, potential and possible restrictions and constraints should be conducted and evaluated in experimental prototypes." (pg.4) "…extended design space is much less defined and understood than the conventional desktop GUI paradigm, requiring more exploratory design approaches." (pg.2) "…MR functionality often requires the use of non-standard hardware… different to the development of conventional desktop GUIs or websites where reasonable generic assumptions with regards to the available displays and interaction devices can be made." (pg.1) "At the beginning of a project, there usually is only a rough idea of the (technical) solution to be developed. This requires that the design space is explored for possible solutions from the user as well as the technology perspective." (pg.3)

Ungrouped quotes		Product Owner Role is divided into three: (pg.5) 1) Platform owner 2) Product leader 3) Feature leader		"…the process was well received by the participants as it matches well with established practices in the different domains like iterative design and rapid prototyping while providing a framework to organize a largely exploratory process." (pg.8) "Using only small extensions and modifications of the Scrum process we were able to integrate these activities into the iterative Scrum structure. The main benefit of this approach is that the same structure is used throughout the whole development." (pg.7)
Observation	Data from four teams developing a student project are presented as showing higher productivity and quality in Agile teams. Paper does not describe how faults were identified/counted.		It is not enough just to apply Agile software development to embedded software. Embedded software has many dependencies and the whole organization needs to support Agility.	Vague and ambiguous requirements call for exploring. It is difficult to manage the exploratory projects using traditional methods. Agile processes relying on feedback, are more suitable. It makes the system development follow the same framework. A collaborative cross-discipline problem and solution space exploration resulted in significantly different design than existing approaches.

Summary of themes in selected studies (C)				
ID	48	42		
Author(s)	Smith, Preston G	Allen, Jacob N., Abdel-Aty-Zohdy, Hoda S. Dr., Ewing, Robert L. Dr.		
Title	Change: Embrace It, Don't Deny It	Agile Hardware Development with Rapid Hardware Definition Language		
Type of study	Technical paper	Technical paper		
Year of pub.	2008	2009		
Agile	Generic	Generic		
Discipline	Generic NPD	Hardware		
Co-design				
Testing		"…allowing for hardware test suites to be developed." (pg.1)		
Iterative HW -technology	"This [Agile development originating from software] does not mean that other fields cannot be Agile, but it does mean that other developers and managers wishing to become more Agile will have to rethink the basics of agility and find other tools and approaches for restoring flexibility to non-software development." (pg.2) "…technology – both the technology that goes into the product and the technology (like computer-aided design tools) used to develop it – is changing at an accelerating pace." (pg.1) "…cost-benefit equation has shifted enormously in recent years as computer-aided technologies have greatly reduced the cost of experimentation in many fields…" (pg.4)	"…hardware can be constructed using any language supported by Microsoft .Net, including C#, Vb.Net, Ruby, F#, Cobol, and others." (pg.1)		

	"Apply them [flexibility tools and approaches] selectively to only the parts of projects where you anticipate change or to only projects facing the prospect of great change." (pg.2) "…one should apply modular architectures selectively where they will contribute the most to flexibility without incurring undue penalties." (pg.4)			
Ungrouped quotes	9 tools and practices: *Continually monitor customers *Fence in change *Try things out *Explore the design space *Build strong teams *Make decisions at the last responsible moment *Plan piecemeal and constantly consider risk *Maintain flexibility in upper layers of process *Out-innovate the competition			
Observations	Paper discusses that learnings from Agile software development can be adapted to generic NPD. NPD in general is facing the challenges of growing change.	Describes a practice that could bring hardware development closer to software development, and thus enable more usage of Agile software development practices.		

APPENDIX B - Interview Instrument A

(Have you experienced problems in projects in the past?)
Can you name the biggest problems?
- Can you describe the reasons?
- Have you tried to fix them?
- How/by whom did you try to fix them?
- Have you been successful in fixing them?

(Did you notice any change in working on the SPS project compared to past projects)?
What kind of change did you notice?
- What was the biggest difference to normal?
- Did your feelings about Agile evolve during the project?
- What do you feel about the feedback loops?
- How much empowerment is executed in your opinion?
- In your opinion, is there any non-value work?
- What do you think about the formal education?
- What do you think about coaching?
- What do you think are the main practices in Agile methods?
- What do you think are the main values in Agile methods?

In contrast to the earlier-mentioned problems, what do you think about Agile methods?
- (probe for all applicable problems earlier mentioned)
- Feedback loops
- Empowerment
- Non-value work

If given a choice, would you prefer to work on an Agile project again?
-Can you explain why you said yes/no?
- Is there anything that should be removed/fixed?

APPENDIX C - Interview Instrument B

Interview Instrument for Project Lead

How was this project started from your perspective?
What was the definition of your role?
How was the project defined to you?
How was the project organization set up?
By whom?
Why that way in your own opinion?

Can you tell me about the reasons and justification for choosing to use Agile methods?
Were there any specific reasons?
Did you have any doubts?
Did you have alternatives for using Agile?
What was your previous experience of Agile?

What happened in the beginning?
How were the teams bootstrapped?
What were the early results?
Did you notice any issues at the beginning?
What were they?
Did the organizations evolve?
Were there any new practices?
Can you explain the cause of them?

How did Sprints in general work?
Planning?
Reviews?
Retrospectives?
Changes in patterns?
How the teams worked together?
How you worked with the customer role?
How did you work with Europe (non-Agile)?

Can you describe what you experienced in the middle section of the project?
Were there any specific incidents that you remember?
How did you see the project going?
Had you made any changes from the beginning of the project?
Were there any major impediments?
How did Sprints in general work?
Planning?
Reviews?
Retrospectives?
Changes in patterns?

How the teams worked together?
How you worked with the customer role?
How did you work with Europe (non-Agile)?

Can you describe what you experienced in the latter part of the project?
Were there any specific incidents that you remember?
How did you see the project going?
Had you made any changes from the beginning of the project?
Were there any major impediments?
How did Sprints in general work?
Planning?
Reviews?
Retrospectives?
Changes in patterns?
How the teams worked together?
How you worked with the customer role?
How did you work with Europe (non-Agile)?

Planning for the last 3 Sprints was done in Grenoble, with very large features and giving a single estimation combining 2 teams. Can you remember how that worked out?
(Reminder) There was a change in mission and scope?
(Reminder) You were planning in the middle with estimates for each team, what do you think about this?

How you think you succeeded in general?
Were you disappointed in anything?
Were you happily surprised by anything?

Were there any new kinds of problems during the whole course of the project that you haven't mentioned?
Did you expect them?
What were they?
Were you able to solve them?
How?
What was the cause of these problems?
Was it because of the introduction of Scrum/Agile?

Specifics: pin-pointing (if not accessed earlier):
Customer role?
Did it evolve?
How did you need to adapt the method?
Do you think this changed the way of working in entities?
How?
How did you see the role of the electronic communication tool?
How did the requirements evolve?

How did you try to solve the knowledge transfer?
How did things go with the Dimmer Specialist in Finland (Raimo)?
Can you explain the review technique when you were developing physical devices instead of software?
How did you present it in reviews?
Can you explain terms like
first-level prototype
functioning prototype
Europe as non-Agile, how did that work out?

Any final recommendations for people who are willing to try a similar approach?

Interview Instrument for Scrum Masters

How were you first introduced to the process to be used in the SPS project?
How long had your organization existed before the beginning of the SPS project?
How different was the approach in SPS?

What happened in the beginning?
How were the teams bootstrapped?
What were the early results?
Did you notice any issues at the beginning?
What were they?
Did the organization evolve?
Were there any new practices?
Can you explain the cause of them?

How did Sprints in general work? (Any changes toward whole team activity?)
Planning?
Reviews?
Retrospectives?
Changes in patterns?
How the teams worked together?
How you worked with the customer role?
How did you work with Europe (non-Agile)?

Can you describe what you experienced in the middle section of the project?
Were there any specific incidents that you remember?
How did you see the project going?
Had you made any changes from the beginning of the project?
Were there any major impediments?
How did Sprints in general work?
Planning?
Reviews?
Retrospectives?
Changes in patterns?
How the teams worked together?
How you worked with the customer role?
How did you work with Europe (non-Agile)?
Was there a change in key personnel in the Chinese team?

Can you describe what you experienced in the latter part of the project?
Were there any specific incidents that you remember?
How did you see the project going?
Had you made any changes from the beginning of the project?
Were there any major impediments?
How did Sprints in general work?

Planning?
Reviews?
Retrospectives?
Changes in patterns?
How the teams worked together?
How you worked with the customer role?
How did you work with Europe (non-Agile)?
China took the responsibility for prototyping. How did this work?

Planning for the last 3 Sprints was done in Grenoble, with very large features and giving a single estimation combining 2 teams. Can you remember how that worked out?
(Reminder) There was a change in mission and scope?
(Reminder) You were planning in the middle with estimates for each team, what do you think about this?

How do you think you succeeded in general?
Were you disappointed in anything?
Were you happily surprised by anything?
Do you think the approach was right?
Can you think of any drivers for this failure/success?
Which ones?

What did you see as the main differences to other approaches?
What was the typical way of communicating with your other "customers"?

What do you see as the main difficulties that were encountered during the project?
Did you manage to solve them?
Yes, How?
No, Why?
Did you have difficulties with legitimacy processes?
Did you have issues with your CMMI audits?

Have you been using similar approaches since SPS?
Have you made any kind of changes since SPS?
What kind?

Specifics: pin-pointing (if not accessed earlier):
Communication
What do you think about the amount of communication using written documents?
How did the electrical tools work for you?
[Case Company] tools?
Why did something work, or not work?
Anything else specific to communication?

Can you explain terms like
first-level prototype
functioning prototype
“IT-flow” (was seen as a problem in Retrospective in Grenoble)
Dispersed teams?
When?
How?
Who?
How did it work?
Pair design?
Other rules
=S=
CMMI
OCP

Engineers working in Europe?
What do you see as the main factor for trust building?
Not all team members met face-to-face?

Would you like to work this way in the future?

Any final recommendations for people who are willing to try a similar approach?